LSH-IS: Un nuevo algoritmo de selección de instancias de complejidad lineal para grandes conjuntos de datos

Álvar Arnaiz-González, José F. Diez-Pastor, César García-Osorio, and Juan J. Rodríguez

Universidad de Burgos, España
{alvarag,jfdpastor,cgosorio,jjrodriguez}@ubu.es

Resumen En las últimas décadas, el crecimiento de las bases de datos se ha incrementado de manera destacable, lo que plantea nuevos retos debido a que los métodos de aprendizaje no están preparados para manejar tales volúmenes de información. La selección de instancias se presenta como una posible aproximación a este problema, no obstante, estos métodos sufren de problemas similares a medida que aumentan los conjuntos de datos.
En este artículo se presenta un nuevo algoritmo de selección de instancias de complejidad lineal que hace uso de las ideas del *Locality-sensitive hashing* para encontrar similitudes entre las instancias. Mientras la complejidad de los métodos tradicionales (de orden cuadrática o linearítmica $\mathcal{O}(n \log n)$) los hace incapaces de manejar grandes volúmenes de información, la propuesta planteada demuestra resultados competitivos en cuanto a precisión y más que destacables en cuanto a tiempo de ejecución. Para comprobar su funcionamiento se ha comparado con algunos de los métodos de selección de instancias más conocidos y se ha evaluado también sobre conjuntos de datos de gran tamaño (hasta un millón de instancias).

Keywords: clasificación, selección de instancias, hashing, LSH

1 Introducción

El clasificador de los k vecinos más cercanos (kNN) [9] se sigue empleando en muchos problemas de aprendizaje automático pese a su antigüedad. Su simplicidad y facilidad de implementación, así como su buen funcionamiento en numerosos dominios, le permiten seguir siendo utilizado pese a sus defectos [19]. El hecho de almacenar todas las instancias de entrenamiento exige una gran cantidad de memoria y tiempo de procesamiento en la tarea de clasificación [22]. Para acelerar el proceso, tradicionalmente se han seguido dos vías: acelerar el cálculo de los vecinos más cercanos o disminuir el tamaño del conjunto de entrenamiento (que es almacenado por el algoritmo).

La reducción del tamaño se puede conseguir disminuyendo el número de atributos (selección de características) o disminuyendo el número de instancias,

ejemplos o prototipos (selección de instancias). El problema que se plantea en este punto es que los mejores algoritmos tradicionales de selección de instancias tienen una complejidad computacional de, como mínimo, $\mathcal{O}(n \log n)$ y muchos incluso mayor. Para superar esta limitación han surgido diversas aproximaciones que decrementan la complejidad mediante remuestreo [13,14].

El artículo se organiza del siguiente modo: en la Sección 2 se presenta la selección de instancias prestando especial interés en los métodos contra los que se ha comparado el algoritmo propuesto, el cual se presenta en la Sección 3; la Sección 4 presenta y analiza los resultados de la experimentación llevada a cabo y, por último, la Sección 5 recoge las conclusiones y líneas de trabajo futuras.

2 Selección de Instancias

En la vida real los conjuntos de datos que se manejan son cada vez más grandes, por lo que muchos sistemas tienen dificultades relacionadas con el tiempo de ejecución y almacenamiento. Por ello, procesar la información disponible puede llegar a convertirse en una tarea difícil e, incluso, inabordable para los algoritmos actuales de clasificación [16]. Cuando se desea utilizar algún algoritmo de aprendizaje basado en instancias, como puede ser la regla del vecino más cercano, este problema puede ser determinante e impedir su ejecución.

En la selección de instancias se agrupan toda una serie de procedimientos y algoritmos que tienen como finalidad la selección de un subconjunto representativo del conjunto de entrenamiento inicial que mantenga su capacidad predictiva [20]. Existen numerosos métodos de selección de instancias para clasificación y siguen apareciendo nuevos (p. ej. [19]). Una completa revisión puede consultarse en [12]. A lo largo de esta sección se detallan algunos de los métodos más representativos que han sido utilizados en la experimentación.

2.1 Algoritmo Condensado de Hart (CNN)

El algoritmo de condensado de Hart o CNN (*Condensed Nearest Neighbor*) [15] se considera la primera propuesta formal de condensado (selección de instancias) para la regla del vecino más cercano. Este método comienza con un conjunto vacío S al que se añade, de manera aleatoria, una instancia correspondiente a cada una de las distintas clases. Posteriormente, cada objeto en el conjunto de entrenamiento es clasificado empleando únicamente los objetos de S; cuando un objeto es clasificado erróneamente, éste se añade a S para garantizar que en el futuro será clasificado correctamente, tras el añadido de la nueva instancia se comienza de nuevo a recorrer todo el conjunto de entrenamiento. El proceso se repite hasta que no existan instancias que sean clasificadas erróneamente.

2.2 Algoritmo Subconjunto Selectivo Modificado (MSS)

En [4] los autores proponen una modificación en la definición de subconjunto selectivo [24] para obtener una mejor aproximación de las fronteras de decisión.

De esta manera, introducen un algoritmo voraz que intenta obtener prototipos selectivos de tal forma que se da preferencia a instancias del conjunto de entrenamiento que están cerca de la frontera de decisión del vecino más cercano original. Este algoritmo constituye una alternativa eficiente al algoritmo del subconjunto selectivo [24] y normalmente es capaz de seleccionar mejores instancias (más cerca de las fronteras).

2.3 Algoritmos Decremental Reduction Optimization Procedure (DROP)

La familia de algoritmos DROP (Decremental Reduction Optimization Procedure) [25] contiene algunos de los métodos que mejores resultados ofrecen en selección de instancias [6,21,23]. El criterio de eliminación de las instancias se define en términos de los vecinos más cercanos y los asociados. La relación de asociado es la inversa a la de vecino más cercano, es decir, una instancia **p** que tiene a **q** como uno de sus vecinos más cercanos es llamada asociada de **q**. El conjunto de los vecinos más cercanos de una instancia es su *vecindario*, todos los asociados de una instancia constituyen el *conjunto de socios* de esa instancia.

Una vez realizados los cálculos de los conjuntos *vecindario* y *conjunto de socios*, el algoritmo comienza recorriendo cada una de las instancias y calculando las variables *with* y *without*. La variable *with* recoge la cantidad de asociados que se clasifican correctamente teniendo a dicha instancia como vecino; el valor de *without* se calcula del mismo modo con la única diferencia que en este caso no se cuenta a la instancia actual como vecino. Si el valor de *without* es mayor o igual que *with* quiere decir que la instancia no aporta información sobre el conjunto de datos, por lo que puede ser eliminada del conjunto solución. En caso de ser eliminada, sus asociados deberán buscar un nuevo vecino. Algunos algoritmos de la familia, como el DROP3, realizan un filtrado de ruido inicial [26].

2.4 Algoritmo Iterative Case Filtering (ICF)

La regla de selección del algoritmo Iterative Case Filtering o ICF [5,7] se basa en los conceptos de *alcance* y *cobertura* que corresponden con *vecindario* y *conjunto de socios* respectivamente del DROP. La regla de selección consiste en eliminar todas aquellas instancias tales que la cardinalidad del conjunto *alcance* sea superior al de *cobertura*. Esto quiere decir que se eliminará una instancia cuando mediante otro objeto se generaliza la información que pudiese proporcionar. Para evitar el efecto nocivo del ruido, ICF comienza con un filtrado al igual que DROP3.

El algoritmo ICF, tras la eliminación del ruido, comienza calculando los conjuntos `coverage` y `reachable` para cada instancia del conjunto original. Tras esto recorre cada instancia comprobando si la cardinalidad de `reachable` es mayor que la cardinalidad de `coverage`, en caso afirmativo se marca dicha instancia para eliminar y se analizan las cardinalidades de la siguiente instancia. El proceso sigue hasta que todas hayan sido recorridas.

3 Locality-sensitive Hashing

El *locality-sensitive hashing* (*LSH*) es un método eficiente para comprobar la similaridad entre elementos, para ello utiliza un conjunto de funciones de *hash* que, al contrario que las utilizadas en otras aplicaciones que buscan la dispersión de los elementos, tienen la particularidad de asignar elementos similares a la misma cubeta con una alta probabilidad. Del mismo modo la probabilidad de que elementos diferentes sean asignados a la misma cubeta es muy baja [18].

Es común utilizar LSH para aumentar la eficiencia del cálculo de vecinos más cercanos [2,12], con lo que un beneficio indirecto que pueden obtener los algoritmos de selección de instancias de LSH es su utilización en los cálculos del vecino más cercano que utilizan muchos de ellos. La utilización de esta técnica permite acelerar los algoritmos de selección de instancias y, en algunos de ellos, puede disminuir su complejidad computacional aunque en ningún caso llega a ser lineal como el método propuesto.

Lo que nosotros proponemos en este artículo es una novedosa utilización de LSH, no como mero soporte del cálculo de vecinos más cercanos, sino como una operación que define el funcionamiento del nuevo algoritmo de selección de instancias. Básicamente, la idea es seleccionar una sola instancia de cada clase en cada una de las cubetas a las que LSH asocia cada instancia. Esto permite hacer la selección de instancias mediante un único recorrido del conjunto de datos, dando lugar por tanto a una complejidad lineal. Surge una pregunta razonable, ¿conseguirá este proceso de selección un subconjunto de instancias que no degrade las capacidades de los algoritmos de clasificación que utilicen dicho subconjunto?. En este artículo se da respuesta a esta pregunta de forma experimental[1]. Pero antes se da una breve introducción a la teoría subyacente en LSH.

3.1 Teoría de las Funciones Localmente Sensibles

En esta sección se sigue [18] para definir formalmente el concepto de sensibilidad local y el proceso de amplificación de familias de funciones localmente sensibles.

Dado un conjunto de objetos S y una medida de distancia D, una familia de funciones de hash $\mathcal{H} = \{h : S \to U\}$ se dice que es (d_1, d_2, p_1, p_2)-sensible si para todo x, y en S se verifica que:

- Si $D(x, y) \leq d_1$, entonces la probabilidad para todo h en $\mathcal{H}$ de que $h(x) = h(y)$ es al menos p_1.
- Si $D(x, y) > d_2$, entonces la probabilidad para todo h en $\mathcal{H}$ de que $h(x) = h(y)$ es como mucho p_2.

[1] En todo caso, adviértase que incluso una cierta degradación de los clasificadores sería aceptable si se consigue una sustancial reducción del almacenamiento o aceleración en la ejecución [8]. Con frecuencia es mejor tener una aproximación rápida que una solución óptima pero cuando ya es demasiado tarde para poder utilizarla.

Nada se dice en esta definición de lo que ocurre cuando la distancia de los objetos está entre d_1 y d_2, no obstante es posible hacer las distancias d_1 y d_2 tan próximas como se quiera, aunque será a costa de que p_1 y p_2 se aproximen igualmente. Pero, como se verá a continuación, es posible combinar familias de funciones de hash que separen p_1 y p_2 sin modificar las distancias d_1 y d_2.

Dada una familia $\mathcal{H}$ de funciones (d_1, d_2, p_1, p_2)-sensible se puede construir una nueva familia $\mathcal{H}'$ mediante las siguientes operaciones de amplificación:

Construcción-Y Las funciones h de $\mathcal{H}'$ se obtienen combinando un número fijo r de funciones $\{h_1, h_2, \ldots, h_r\}$ de $\mathcal{H}$ de modo que $h(x) = h(y)$ si y solo si $h_i(x) = h_i(y)$ **para todo** i. Si se puede garantizar la independencia en la elección de las funciones de $\mathcal{H}$ para la construcción de las funciones de $\mathcal{H}'$, la nueva familia de funciones será $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensible.

Construcción-O Las funciones h de $\mathcal{H}'$ se obtienen combinando un número fijo b de funciones $\{h_1, h_2, \ldots, h_b\}$ de $\mathcal{H}$ de modo que $h(x) = h(y)$ si y solo si $h_i(x) = h_i(y)$ **para algún** i. Si se puede garantizar la independencia en la elección de las funciones de $\mathcal{H}$ para la construcción de las funciones de $\mathcal{H}'$, la nueva familia de funciones será $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensible.

La construcción-Y disminuye las probabilidades y la construcción-O las aumenta, pero con una adecuada elección de r y b, y el encadenamiento de construcciones, se puede conseguir que la probabilidad p_1 se aproxime a 1, mientras que p_2 se mantenga razonablemente cerca de 0.

Sólo queda definir cómo se van a obtener las funciones de hash de la familia de partida. Esta decisión depende de cómo se vaya a medir la distancia. En la comprobación experimental se ha utilizado la biblioteca TarsosLSH[2], que constituye las funciones de la familia base utilizando la ecuación [10].

$$h_{\boldsymbol{a},b}(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{x} + b}{w} \right\rfloor$$

donde $\boldsymbol{a}$ es un vector aleatorio, b un número real elegido aleatoriamente en el intervalo $[0, w]$ y w la anchura de cada cubeta en las que se aplicarán las instancias.

El funcionamiento se define en el Algoritmo 1, el cual recibe un conjunto de entrenamiento y una familia de funciones de hash. El algoritmo recorre cada instancia del conjunto original X, aplica las funciones hash de la familia y recorre cada cubeta u a la que haya sido asignada la instancia $\mathbf{x}$. Si en la cubeta u no existe ninguna instancia de la misma clase que $\mathbf{x}$, se añade dicha instancia al conjunto editado. Esta forma de utilizar las funciones de la familia de funciones de hash estaría en la línea de lo que se hace con la construcción-O. El algoritmo finaliza cuando todas las instancias han sido recorridas.

4 Configuración Experimental

La experimentación se ha realizado en Weka [27] implementando el nuevo algoritmo (LSH-IS) como filtro con la siguiente configuración: 10 funciones para la

[2] Disponible en: `https://github.com/JorenSix/TarsosLSH`

Algoritmo 1: LSH-IS – Algoritmo de selección de instancias mediante hashing.

Input: Conjunto de instancias $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, conjunto $\mathcal{G}$ de familias de funciones hash
Output: Conjunto de instancias seleccionado $S \subset X$

```
1 S = ∅
2 foreach instancia x ∈ X do
3     foreach familia de funciones g ∈ G do
4         u ← cubeta asignada por la familia g a la instancia x
5         if no existen otras instancias de la misma clase que x en u then
6             Añadir x a S
7             Añadir x a u
8 return S
```

construcción-Y ($r = 10$) y 4 funciones para la construcción-O ($b = 4$) (explicado en la Sección 3.1). El algoritmo propuesto se ha comparado frente a cuatro conocidos métodos de selección de instancias: CNN, ICF, MSS y DROP3, la implementación de éstos ha sido la presentada en [3]. El número de vecinos cercanos utilizados en ICF y DROP3 ha sido $k = 3$.

La ejecución se ha realizado sobre un servidor Cluster Dell con 63 nodos 4xQuadCore Xeon @ 2 GHz (en total 252 núcleos de procesamiento), sistema operativo CentOS, configurado con un sistema de gestión de colas de trabajo PBS (Portable Batch System).

Los experimentos se han realizado mediante validación cruzada con 10 grupos sobre 30 conjuntos de datos disponibles en el repositorio de Keel [1] que aparecen recogidos en la Tabla 1 (se han escogido aquellos con, al menos, mil instancias). La única transformación realizada a los conjuntos de datos ha sido la normalización de todos sus atributos de entrada (entre 0 y 1).

Una vez ejecutados los experimentos, se ha calculado el ranking medio sobre la precisión (% de aciertos del vecino más cercano) (ver Tabla 2). El mejor método es MSS aunque la diferencia respecto a los demás no es significativa según el procedimiento de Hochberg [17] al 95 % de confianza.

La Tabla 3 muestra el ranking medio sobre el tiempo de ejecución, donde el método propuesto supera a todos los demás de manera significativa. La relación entre el tiempo de ejecución y el tamaño del conjunto de datos para los algoritmos analizados aparece en la Figura 1. El gráfico muestra las curvas de Bezier obtenidas con los tiempos medidos en la experimentación. Mientras que métodos como CNN y DROP3 crecen muy rápidamente, MSS e ICF crecen a un ritmo más suave aunque mucho más rápido que el nuevo método.

Puesto que los métodos de selección de instancias tradicionales no están preparados para conjuntos masivos, en la experimentación realizada sobre Census, KDDCup99 y Poker (ver Tabla 1) se ha comparado el tiempo de test del clasificador del vecino más cercano entrenado con el conjunto de datos original y

Tabla 1. Conjuntos de datos utilizados en la experimentación. Se detallan los atributos desglosados en numéricos (C) y nominales (N), el número de instancias (# inst.) y la precisión (% de aciertos del vecino más cercano). Los conjuntos de datos bajo la línea punteada son conjuntos muy grandes que no han podido ejecutarse con los algoritmos tradicionales.

	Datasets	# atrib.		# inst.	Precisión
		C.	N.		
1	German	7	13	1 000	72.90
2	Flare	0	11	1 066	73.26
3	Contraceptive	9	0	1 473	42.97
4	Yeast	8	0	1 484	52.22
5	Wine-quality-r	11	0	1 599	64.85
6	Car	0	6	1 728	93.52
7	Titanic	3	0	2 201	79.06
8	Segment	19	0	2 310	97.23
9	Splice	0	60	3 190	74.86
10	Chess	0	35	3 196	72.12
11	Abalone	7	1	4 174	19.84
12	Spam	0	57	4 597	91.04
13	Wine-quality-w	11	0	4 898	65.40
14	Banana	2	0	5 300	87.21
15	Phoneme	5	0	5 404	90.19
16	Page-blocks	10	0	5 472	95.91
17	Texture	40	0	5 500	99.04
18	Optdigits	63	0	5 620	98.61
19	Mushroom	0	22	5 644	100.00
20	Satimage	37	0	6 435	90.18
21	Marketing	13	0	6 876	28.74
22	Thyroid	21	0	7 200	92.35
23	Ring	20	0	7 400	75.11
24	Twonorm	20	0	7 400	94.81
25	Coil 2000	85	0	9 822	90.62
26	Penbased	16	0	10 992	99.39
27	Nursery	0	8	12 960	98.13
28	Magic	10	0	19 020	80.95
29	Letter	16	0	20 000	96.04
30	KR vs. K	0	6	28 058	73.05
31	Census	7	30	299 285	93.01
32	KDDCup99	54	0	494 021	99.96
33	Poker	5	5	1 025 010	50.62

Tabla 2. Ranking medio sobre la precisión de los métodos comparados y p valor del procedimiento de Hochberg.

Algoritmo	Ranking	p
MSS	2.6167	
DROP3	2.7334	0.7750
CNN	3.0000	0.6955
LSH-IS	3.3000	0.2825
ICF	3.3500	0.2825

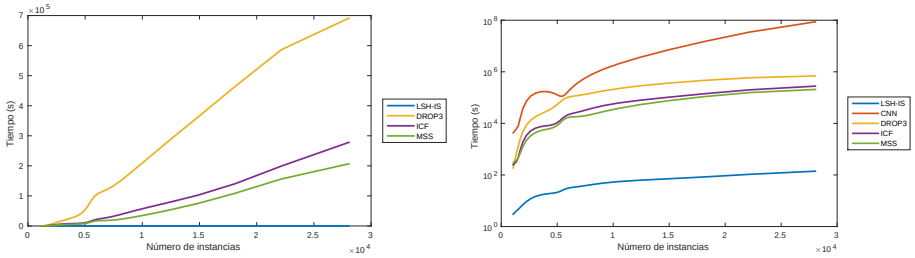
Tabla 3. Ranking medio sobre el tiempo de filtrado de los métodos comparados y p valor del procedimiento de Hochberg.

Algoritmo	Ranking	p
LSH-IS	1.0000	
MSS	2.5833	0.00010
ICF	2.8167	0.00001
DROP3	4.0000	0.00000
CNN	4.6000	0.00000

Tabla 4. Resultados sobre los conjuntos de datos grandes: precisión y tiempo de test para el vecino más cercano con el conjunto de datos original y el subconjunto seleccionado por el algoritmo LSH-IS.

Datasets	Original		Subconjunto LSH-IS	
	Precisión	Tiempo (s)	Precisión	Tiempo (s)
Census	93.01	163.85	91.88	91.48
KDDCup99	99.96	4 011.77	99.50	9.82
Poker	50.62	7 360.18	45.60	3 259.12

con el subconjunto seleccionado por nuestro algoritmo. En la Tabla 4 se observa como el tiempo de test (clasificación), utilizando el subconjunto seleccionado, es mucho menor que utilizando el conjunto original. Este tiempo se ha conseguido a costa de que la precisión disminuya ligeramente en Census y KDDCup99 pero de manera más destacada en Poker.



(a) Lineal (b) Logarítmica

Figura 1. Curvas realizadas aplicando la interpolación de Bezier sobre el tiempo de filtrado con escala lineal (a) y logarítmica (b) en el eje y. En (a) el algoritmo CNN no se ha representado debido a que su crecimiento es tan pronunciado que impedía apreciar las diferencias entre los otros métodos al "*aplanar*" sus curvas.

5 Conclusiones y Líneas Futuras

Se ha presentado un nuevo algoritmo de selección de instancias que permite el procesamiento de conjuntos de datos en una sola pasada, consiguiendo por tanto una complejidad lineal y haciéndolo adecuado para el procesamiento de grandes conjuntos de datos, así como para el procesamiento de datos obtenidos en *stream* sin que sea necesario almacenar la totalidad de éstos en memoria. Para ello utiliza las técnicas de LSH de una forma novedosa para determinar qué instancias se descartan y cuáles se mantienen. Experimentalmente se ha comprobado que ofrece unos resultados competitivos consiguiendo una buena precisión en clasificadores entrenados con el conjunto de datos seleccionado.

En la versión actual, el algoritmo presentado es muy "agresivo" en la forma que hace la selección de las instancias. Se podría mejorar la selección obtenida si la información almacenada en las cubetas fuera algo más elaborada que simplemente la clase de la instancia y el hecho de que ya se ha producido una selección. Se espera que almacenando información adicional, por ejemplo la media incremental de las instancias aplicadas a la cubeta, o estadísticas del porcentaje de instancias de cada clase, o una muestra de más de una instancia, se pueda tomar una decisión más informada sin penalizar excesivamente el tiempo de ejecución. La información utilizada podría ser incluso mayor si se sacrifica el procesamiento en *stream* y se permite un segundo "barrido" del conjunto de datos en el que se utilizaría información recopilada en el primer "barrido" (conviene destacar

que aunque se aumentaría el tiempo de ejecución, la complejidad del algoritmo seguiría siendo lineal).

En la experimentación, el método propuesto se ha enfrentado contra cuatro algoritmos "clásicos" del estado del arte. Una de las próximas tareas es la de comparar su funcionamiento contra otros métodos más novedosos que han ido surgiendo en los últimos años [19]

La aplicación de este método para Big Data es una de las líneas futuras más inmediatas. Su adaptación a una filosofía *map-reduce* [11] permitiría su implementación en plataformas de Big Data como, por ejemplo, *Spark*. Aunque en el artículo se ha utilizado el vecino más cercano, el uso de otros clasificadores más adecuados para Big Data es una de las tareas pendientes.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad, proyecto TIN 2011-24046.

Referencias

1. J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, and F. Herrera. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.
2. Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. *CoRR*, abs/1501.01062, 2015.
3. Álvar Arnaiz-González, José-Francisco Díez-Pastor, César García-Osorio, and Juan-José Rodríguez Díez. Herramienta de apoyo a la docencia de algoritmos de selección de instancias. In *Jornadas de Enseñanza Universitaria de la Informática (JENUI)*, pages 33–40. Universidad de Castilla La Mancha, 2012.
4. Ricardo Barandela, Francesc J. Ferri, and José Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *IJPRAI*, 19(6):787–806, 2005.
5. Henry Brighton and Chris Mellish. On the consistency of information filters for lazy learning algorithms. In JanM. Żytkow and Jan Rauch, editors, *Principles of Data Mining and Knowledge Discovery*, volume 1704 of *Lecture Notes in Computer Science*, pages 283–288. Springer Berlin Heidelberg, 1999.
6. Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6:153–172, 2002.
7. Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, 2002.
8. Jorge Calvo-Zaragoza, Jose J. Valero-Mas, and Juan R. Rico-Juan. Improving knn multi-label classification in prototype selection scenarios using class proposals. *Pattern Recognition*, 48(5):1608 – 1622, 2015.
9. T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967.

10. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.
11. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
12. S. Garcia, J. Derrac, J.R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):417–435, March 2012.
13. César García-Osorio, Aida de Haro-García, and Nicolás García-Pedrajas. Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artificial Intelligence*, 174(5–6):410 – 441, 2010.
14. Nicolás García-Pedrajas and Aida de Haro-García. Boosting instance selection algorithms. *Knowledge-Based Systems*, 67(0):342 – 360, 2014.
15. P. Hart. The condensed nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 14(3):515 – 516, may 1968.
16. Pablo Hernandez-Leal, J. Ariel Carrasco-Ochoa, J.Fco. Martínez-Trinidad, and J. Arturo Olvera-Lopez. Instancerank based on borders for instance selection. *Pattern Recognition*, 46(1):365 – 375, 2013.
17. Yosef Hochberg. A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):800–802, 1988.
18. Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014.
19. Enrique Leyva, Antonio González, and Raúl Pérez. Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. *Pattern Recognition*, 48(4):1523 – 1537, 2015.
20. Loris Nanni and Alessandra Lumini. Prototype reduction techniques: A comparison among different approaches. *Expert Systems with Applications*, 38(9):11820 – 11828, 2011.
21. José Arturo Olvera-López, J Fco. Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, and Josef Kittler. Prototype selection based on sequential search. *Intelligent Data Analysis*, 13(4):599–631, 2009.
22. Stefanos Ougiaroglou, Georgios Evangelidis, and Dimitris A Dervos. FHC: an adaptive fast hybrid method for k-NN classification. *Logic Journal of the IGPL*, 2015.
23. JA Pérez-Benítez, JL Pérez-Benítez, and JH Espina-Hernández. Novel data condensing method using a prototype's front propagation algorithm. *Engineering Applications of Artificial Intelligence*, 39:181–197, 2015.
24. GL Ritter, HB Woodruff, SR Lowry, and TL Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.
25. D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pages 404–411. Morgan Kaufmann, 1997.
26. Dennis L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-2(3):408–421, July 1972.
27. Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.