Selección de Prototipos Basada en Conjuntos Rugosos Difusos

Nele Verbiest[1], **Chris Cornelis**[1,2], **Francisco Herrera**[2]

[1] *Universiteit Gent, Bélgica*, {Nele.Verbiest,Chris.Cornelis}@UGent.be
[2] *Universidad de Granada, España*, herrera@decsai.ugr.es

Abstract

En este trabajo abordamos uno de los principales problemas de k vecinos más cercanos (kNN): su sensibilidad al ruido. Llevamos a cabo Selección de Prototipos (SP), es decir, eliminamos instancias ruidosas para mejorar la calidad de la clasificación de k vecinos más cercanos. Concretamente, basándonos en un método existente de selección de instancias basada en conjuntos rugosos difusos, construimos un algoritmo de tipo envoltura que tiene en cuenta la granularidad óptima de la relación difusa de indiscernibilidad en cada conjunto de datos. Llamamos a este método Selección de Prototipos a base de Conjuntos Aproximados Difusos (SPCAD). La comparación del enfoque con el estado del arte en Selección de Prototipos confirma que nuestro método ofrece buenos resultados: supera a todos los métodos de selección de prototipos existentes con respeto a la precisión de la clasificación.

Palabras clave: conjuntos rugosos difusos, kNN, selección de prototipos

1. Introducción

El algoritmo kNN (k vecinos más cercanos ,[10]) es uno de los clasificadores no paramétricos más usados en el campo de minería de datos y de aprendizaje automático. En general, un clasificador dispone de datos de entrenamiento (un sistema de decisión X que consiste en instancias con sus valores de atributos y su clase) e intenta predecir la clase $d(t)$ de una instancia objetivo t. kNN resuelve este problema buscando las k instancias de X más similares a t y asignando t a la clase mayoritaria entre estas k instancias.

El clasificador kNN debe su popularidad al hecho de que es simple, fácil de implementar y presenta un sesgo reducido: para un problema de clasificación con dos clases, la tasa de error de 1NN nunca excederá el doble del error óptimo de Bayes.

Sin embargo, kNN tiene algunas desventajas. Uno de sus problemas es la alta dimensionalidad [4]. La selección de características hace frente a este problema seleccionando solo características relevantes y eliminando características redundantes (p.e. [18]).

En segundo lugar, kNN necesita mucho almacenamiento: el conjunto de entrenamiento completo tiene que ser almacenado para la clasificación de nuevas instancias. Además, kNN necesita mucho cómputo por que hace falta calcular las similitudes entre la instancia nueva y todas las instancias de entrenamiento. Finalmente, kNN es muy susceptible a los datos ruidosos, ya que considera todos los datos igual de relevantes, no teniendo en cuenta que el conjunto de entrenamiento pueda contener datos incorrectos.

La Selección de Prototipos (SP, [14]) hace frente a dichos problemas: reduce el conjunto de entrenamiento para seleccionar los vecinos más cercanos. Se ha realizado mucho trabajo en el campo de métodos de SP; en [15], se presenta una taxonomía extensa de métodos existentes. En primer lugar, podemos distinguir entre los métodos SP basándonos en el tipo de instancias que eliminan. Los métodos de edición quitan instancias ruidosas, mientras que los métodos de condensación principalmente eliminan instancias supérfluas, y los métodos híbridos intentan eliminar ambos tipos de instancias al mismo tiempo.

En segundo lugar, los métodos se agrupan en métodos de filtro y de envoltura. En el contexto de SP, los métodos de envoltura utilizan el conjunto entero de entrenamiento X para evaluar un subconjunto de instancias $S \subseteq X$. La calidad de S se mide clasificando X con una estrategia "leave-one-out": cada instancia x en X se clasifica buscando vecinos dentro de $S \setminus \{x\}$ si $x \in S$ y dentro de S en otro caso. Los métodos filtro solo usan datos parciales para decidir si una instancia dada debe ser retenida o eliminada. Por ejemplo, el método CNN selecciona una instancia si la regla 1NN ejecutada con el subconjunto actual de instancias la cla-

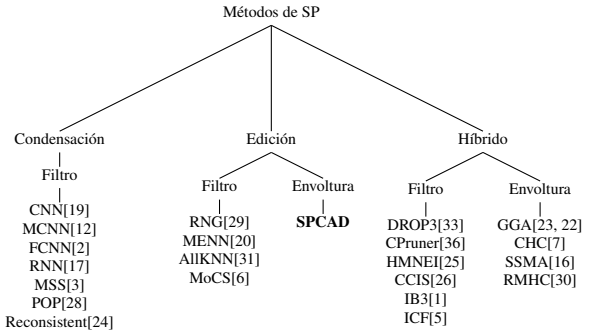

Figura 1: Taxonomía de Métodos de SP, incluido SPCAD

sifica mal. En general, los métodos de envoltura son más exactos pero también más lentos que los filtros. En la Figura 1, se representa la taxonomía de SP. Para cada grupo de métodos, se muestran los que tienen mejor desempeño.

El método que proponemos en este trabajo tiene como objetivo eliminar las instancias ruidosas y lo consigue evaluando la calidad de subconjuntos de instancias sobre el conjunto de entrenamiento completo. Por lo tanto, nuestra propuesta, Selección de Prototipos a base de Conjuntos Rugosos Difusos (SPCAD), es un método de envoltura y de edición. SPCAD utiliza los conjuntos rugosos difusos (CRD, [13]), un modelo híbrido que une la teoría de los conjuntos rugosos [27] que modelan el conocimiento imperfecto y la de los conjuntos difusos [35] que modelan datos imprecisos. La teoría CRD se ha explorado extensamente para la selección de características [9] pero su uso para SP todavía se encuentra en una etapa preliminar. En [21]) se presentó un primer método de selección de instancias basado en la teoría CRD. En este trabajo, extendemos este método y mejoramos su precisión usando un enfoque de envoltura.

En la siguiente sección, describimos SPCAD y en la Sección 3, mostramos su buen desempeño en problemas reales. Concluimos el trabajo en la sección 4.

2. Selección de Prototipos Basada en Conjuntos Rugosos Difusos

En lo sucesivo, se considera un sistema de decisión $(X, \mathscr{A} \cup \{d\})$, donde $X = \{x_1, \ldots, x_n\}$ es un conjunto de instancias, $\mathscr{A}$ es un conjunto de atributos $\{a_1, \ldots, a_m\}$, y d es el atributo de decisión (no incluido en $\mathscr{A}$). El valor de la instancia x para el atributo a se denota por $a(x)$ y puede ser continuo (real) o nominal. El atributo de decisión de una instancia x, $d(x)$, siempre es nominal. Sin perdida de generalidad, asumimos que todos los atributos continuos están normalizados: para cada $x \in X$, $a(x) \in [0,1]$.

El nucleo de la teoría CRD es el concepto de región positiva difusa. Para una instancia $x \in X$, el grado de pertenencia a la región positiva difusa está dado por:

$$POS^{\alpha}_{\mathscr{A}}(x) = \inf_{y \in X} \mathscr{I}(R^{\alpha}_{\mathscr{A}}(x,y), R_d(x,y)).$$

En esta fórmula, $\mathscr{I}$ es una implicación difusa y $R_d(x,y) = 1$ si las instancias x e y tienen la misma decisión y 0 en otro caso. $R^{\alpha}_{\mathscr{A}}(x,y)$ mide la indiscernibilidad de las instancias x e y con respeto a los atributos $\mathscr{A}$. Se define por:

$$R^{\alpha}_{\mathscr{A}}(x,y) = \underbrace{\mathscr{T}(\max(0, 1 - \alpha\delta_a(x,y)))}_{a \in \mathscr{A}},$$

donde $\mathscr{T}$ es una norma triangular (t-norma) y δ_a es una medida de distancia basada en el atributo a. Nótese que esta definición sigue siendo válida incluso si $\mathscr{A}$ contiene más de dos atributos, ya que una t-norma es asociativa y por lo tanto se puede extender a $[0,1]^m \to [0,1]$ inequívocamente. Usamos la siguiente medida de distancia en el caso de un atributo nominal a:

$$\delta_a(x,y) = \begin{cases} 0, & \text{si } a(x) = a(y) \\ 1, & \text{en otro caso} \end{cases}$$

Si a es continuo, usamos:

$$\delta_a(x,y) = (a(x) - a(y))^2.$$

Como asumimos que todos los atributos continuos están normalizados, $R^{\alpha}_{\mathscr{A}}$ producirá un valor entre 0 y 1.

El parámetro $\alpha \in [0, +\infty[$ se llama la *granularidad* y expresa lo grandes que las diferencias entre valores de atributos de instancias tienen que ser para poder distinguirse entre ellas. Cuando α es más pequeño, los valores de atributos de las instancias tendrán que diferenciarse más.

Para un α determinado, la región positiva evalua el grado en el que todas las instancias indiscernibles de una instancia x pertenecen a la misma clase. El algoritmo FRIS (Fuzzy Rough Instance Selection, [21]) emplea esta idea y elimina todas las instancias que no pertenecen completamente a la región positiva difusa ($POS_{\mathscr{A}}(x) < 1$).

FRIS usa un valor fijo para el parámetro de granularidad α. Sin embargo, los experimentos en [21] mostraban que el desempeño de FRIS depende altamente de la elección de α, y que el valor óptimo de α depende del conjunto de datos. Por tanto, SPCAD estima el valor óptimo de α para FRIS usando un enfoque de envoltura. A continuación, escribimos FRIS$^{\alpha}$ para denotar el algoritmo FRIS usando α como parámetro de granularidad.

Antes de introducir SPCAD, consideramos el siguiente teorema demostrable a partir de la teorïa CRD:

Teorema de Granularidad Mínima:
Sea $\mathscr{I}$ una implicación difusa tal que $\forall t \in [0,1]$, $\mathscr{I}(t,0) = 1 - t$ (se mantiene por ejemplo para las conocidas implicaciones de Łukasiewicz y Kleene-Dienes), y sea $x \in X$.

Entonces si $\mathscr{T}$ es el mínimo o el producto:

$$\sup\{\alpha \in [0,+\infty[| POS^{\alpha}_{\mathscr{A}}(x) < 1\} = \max_{y \notin [x]_d} \frac{1}{\max_{i=1}^{m} \delta_{a_i}(x,y)} \quad (1)$$

y si $\mathscr{T}$ es la t-norma de Łukasiewicz:

$$\sup\{\alpha \in [0,+\infty[| POS^{\alpha}_{\mathscr{A}}(x) < 1\} = \max_{y \notin [x]_d} \frac{1}{\sum_{i=1}^{m} \delta_{a_i}(x,y)}, \quad (2)$$

siempre que los denominadores sean distintos de 0.

Nótese que, en caso de que los denominadores son cero, significa que existen $x, y \in X$ tales que $\forall a \in \mathscr{A}, a(x) = a(y)$, pero $d(x) \neq d(y)$. Entonces, para cada $\alpha \in [0,+\infty[$, $POS^{\alpha}_{\mathscr{A}}(x) = POS^{\alpha}_{\mathscr{A}}(y) = 0$. En otras palabras, x e y son eliminadas por $FRIS^{\alpha}$ para cada valor de α.

Denotamos por $\alpha(x)$ el valor crítico de x y usamos la notación simbólica ∞ para instancias que siempre están eliminadas por $FRIS^{\alpha}$, independientemente del valor de α:

$$\forall x \in X : \alpha(x) = \begin{cases} \infty \text{ si } \exists y \notin [x]_d, \forall a \in \mathscr{A}\, a(x) = a(y) \\ \sup\{\alpha \in [0,+\infty[| POS^{\alpha}_{A}(x) < 1\} \text{ en otro caso} \end{cases}$$

Esto significa que si $FRIS^{\alpha}$ se ejecuta con un parámetro de granularidad α menor de $\alpha(x)$, elimina la instancia. Por otro lado, si α es mayor o igual a $\alpha(x)$, $FRIS^{\alpha}$ conserva la instancia. En el caso especial case donde $\alpha(x) = \infty$, $FRIS^{\alpha}$ siempre elimina x, independientemente de la elección de α.

Ahora podemos volver a formular el problema de encontrar un valor óptimo de α para FRIS. Asumimos que las instancias $x \in X$ están ordenadas según sus valores $\alpha(x)$:

$$\alpha(x_1) \leq \alpha(x_2) \ldots \leq \alpha(x_n),$$

donde asumimos que $\infty \leq \infty$ y $\forall r \in \mathbb{R}, r < \infty$.

El algoritmo $FRIS^{\alpha}$ elimina todas las instancias x para las que $\alpha < \alpha(x)$, es decir, existe un *índice de división* $i \in \{1,\ldots,n\}$ tal que las instancias $x_1,\ldots,x_i$ se conservan y que las instancias $x_{i+1},\ldots,x_n$ se eliminan. Esto se reduce a aplicar el algoritmo $FRIS^{\alpha(x_i)}$. Significa que encontrar un buen $\alpha \in [0,\infty[$ para $FRIS^{\alpha}$ es equivalente a encontrar un buen $\alpha \in \{\alpha(x_1),\ldots,\alpha(x_n)\}$ para $FRIS^{\alpha}$.
El algoritmo SPCAD explota esta idea. Evalua los subconjuntos $S = \{x_1,\ldots,x_i\}$ de instancias correspondientes a los algoritmos $FRIS^{\alpha(x_i)}$ $(i = 1,\ldots,n)$ calculando sus precisiones en entrenamiento usando leave-one-out. Selecciona los valores α que proporcionan un acierto de entrenamiento máximo. En caso de que distintos valores de α corresponden al mejor acierto en entrenamiento, tomamos su mediana.

En Algoritmo 1, se muestra el esquema final del algoritmo. En primer lugar, los valores $\alpha(x)$ se calculan para cada $x \in X$. Después, se eliminan los valores duplicados y se ordenan los restantes, teniendo en cuenta de nuevo que

Algorithm 1 SPCAD

```
1: input: Sistema de decisión (X, A ∪ {d})
2: Calcula α(x_1), ..., α(x_n)
3: Elimina duplicados y ordena los valores de α del paso
   2: α_1 > α_2 > ... > α_p
4: opt.alphas ← {∞}
5: Calcula vecinos más cercanos de todas las instancias
6: acc.opt ← accuracy(X, A ∪ {d})
7: acc.current ← acc.opt
8: for α = α_2, ..., α_p do
9:    Elimina instancias x para las que α(x) > α
10:   if número de instancias restantes > 1 then
11:      Recalcula vecinos más cercanos de instancias para las que el vecino más cercano actual es eliminado
12:      Recalcula acc.current
13:      if acc.current > acc.opt then
14:         opt.alphas ← {α}
15:      else if acc.current = acc.opt then
16:         opt.alphas ← opt.alphas ∪ {α}
17:      end if
18:   end if
19: end for
20: best.alpha = median(opt.alphas)
21: Output FRIS^{best.alpha}(X, A ∪ {d})
```

$\forall r \in \mathbb{R}, r < \infty$. Por sencillez, asumimos que $FRIS^{\infty}$ devuelve el conjunto de entrenamiento X entero. En la línea 5, se obtiene el vecino más cercano para cada instancia. A continuación, en la línea 6 se calcula el acierto usando el conjunto entero de entrenamiento. En cada ejecución del bucle, las instancias para las que $\alpha(x)$ excede el umbral actual α se eliminan del conjunto actual de instancias. Nótese que esto equivale a aplicar $FRIS^{\alpha}$. Si el acierto del conjunto de instancias actual es igual o mejor que el mejor acierto encontrado hasta ahora, se actualizan este último valor y la lista correspondiente de valores α optimales. Finalmente, el mejor valor de α se calcula como la mediana de todos los valores óptimos de α values, y se devuelve el subconjunto obtenido con $FRIS^{best.alpha}$. La razón por que consideramos los valores de α en orden decreciente es que así podemos implementar este paso de manera eficaz. Los subconjuntos de instancias son generados de manera decreciente. Cada vez que se eliminan instancias del subconjunto actual, solo recalculamos los vecinos más cercanos de las instancias afectadas por esta operación. El criterio de parada en la línea 10 asegura que se puede obtener un vecino más cercano para cada instancia; si solo hay una instancia x para la que $\alpha(x) = \alpha_p$, es decir, solo hay una instancia con menor valor de α, entonces en la línea 11 el vecino más cercano no se puede obtener para x: el único candidato es x pero esta instancia no se puede elegir como vecino más cercano de x en la estragía leave-one-out. Por eso, optamos por no considerar este subconjunto $\{x\}$.

Un posible inconveniente del método básico SPCAD es que las fórmulas (1) y (2) para $\alpha(x)$ están basadas en el operador máximo. Esto significa que cambios pequeños en los datos pueden alterar los cálculos de manera drástica, y por lo tanto la robustez de la selección de prototipos podría verse afectada. Por tanto, consideramos una generalización del enfoque básico usando operadores de agregación del tipo OWA [34].

Recordemos que, dada una serie de valores $a_1,\ldots,a_p \in \mathbb{R}$ y un vector de pesos $W = \langle w_1,\ldots,w_p\rangle$ tal que $\forall i \in 1,\ldots,p$: $w_i \in [0,1]$ y $\sum_{i=1}^{p} w_i = 1$, la agregación OWA de estos valores se obtiene como $[OWA_W(a_1,\ldots,a_p) = \sum_{i=1}^{p} w_i b_i$, donde $b_i = a_j$ si a_j es el i-ésimo mayor valor entre $a_1,\ldots,a_p$. Es decir, los valores se ordenan y después se extrae un promedio ponderado de estos valores.

El operador OWA se puede usar para relajar la idea del máximo, haciéndolo menos estricto. Consideremos un vector de pesos $W_{max} = \langle w_1,\ldots,w_p\rangle$ tal que $w_1 \geq w_2 \ldots \geq w_p$, donde valores mayores corresponden a pesos mayores. Se puede usar $OWA_{W_{max}}$ para calcular los $\alpha(x)$ de cada $x \in X$. Para $\mathscr{T} = \mathscr{T}_L$, la definición generalizada de $\alpha(x)$ se denota por:

$$\forall x \in X : \alpha^{OWA}(x) = \underbrace{OWA_{W_{max}} \frac{1}{\sum_{i=1}^{m} \delta_{a_i}(x,y)}}_{y \notin [x]_d},$$

mientras que para $\mathscr{T} = \mathscr{T}_M$ or $\mathscr{T} = \mathscr{T}_P$, podemos generalizar la definición de $\alpha(x)$ por:

$$\forall x \in X : \alpha^{OWA}(x) = \underbrace{OWA_{W_{max}} \underbrace{\frac{1}{OWA_{W_{max}} \delta_{a_i}(x,y)}}_{i=1,\ldots,m}}_{y \notin [x]_d}.$$

La ventaja del operador OWA_W es que todos los pesos pueden ser mayores de cero, lo que significa que todos valores pueden influir el resultado de agregación y se pueden resultados más estables. Un posible vector de pesos para una agregación OWA con el comportamiento del máximo puede obtenerse por:

$$\forall i \in 1,\ldots,p : w_i = \frac{2(p-i+1)}{p(p+1)}. \quad (3)$$

3. Resultados Experimentales

3.1. Configuración Experimental

En esta sección estudiamos el rendimiento de SPCAD. Seguimos la configuración experimental del estudio realizado en [15]. Empleamos 58 conjuntos de datos y sus particiones del repositorio KEEL[1] y comparamos SPCAD con los 21 métodos de selección de prototipos que tienen mejor desempeño entre los diferentes tipos de métodos de SP.

Usamos la estrategia de validación cruzada en 10 partes para evaluar el rendimiento de los clasificadores. Para cada parte (datos de test), reducimos las partes restantes (datos de entrenamiento) usando un método de SP determinado, y clasificamos los datos de test con kNN, empleando los datos de entrenamiento reducidos.

El rendimiento de SPCAD se compara usando 4 medidas:

- *Precisión (acc):* la tasa de instancias de test clasificadas correctamente
- *Kappa de Cohen (κ) [8]:* una medida adicional de precisión, basándose en la matriz de confusión, que compensa los éxitos obtenidos al azar, .
- *reducción de almacenamiento (red):* la fracción de instancias eliminadas del conjunto de entrenamiento
- *tiempo de ejecucíon (time):* el tiempo de ejecucíon en segundos del método SP. El tiempo de ejecución de la clasificación 1NN posterior no se considera.

Experimentos[2] muestran que SPCAD utilizando $\mathscr{T}_L$ como t-norma y el enfoque OWA con los pesos definidos como en la ecuación (3) obtienen los mejores resultados. Por lo tanto, empleamos esta configuración para los experimentos. Además, solo experimentamos con el clasificador 1NN.

3.2. Resultados

En la Tabla 1, se muestran los resultados medios para todos los métodos de SP. De dicha tabla podemos concluir que SPCAD funciona muy bien con respeto a exactitud y Cohen's kappa. El test de Wilcoxon [3] [32] confirma esta conclusión: comparando SPCAD con cada de los demás métodos SP con respeto a exactitud y kappa, SPCAD siempre los supera con un nivel de $\alpha = 0{,}1$ de significancia.

Pese a ello, usando el test de Wilcoxon para multiples comparaciones por pares, perdemos el control sobre la verdadera tasa de error, es decir la probabilidad de hacer uno o más discubrimientos falsos entro todas las hipótesis [11]. Por ello, también usamos el test de Friedman y el procedimiento post-hoc de Holm [3] [11], diseñado especificamente para comparar multiples hipótesis. Hemos aplicado este test solo para los 9 algoritmos con mejor desempeño con respeto a la precisión en test y kappa (AllKNN, CHC, GGA, HMNEI, MoCS, RMHC, RNG, SSMA, FRPS) ya que el test pierde poder si se comparan demasiados algoritmos [11].

[1] http://sci2s.ugr.es/keel/datasets.php

[2] No incluidos en este trabajo debido a restricciones de espacio. Se pueden consultar en: users.ugent.be/~nverbies

[3] Las estadísticas de este test se pueden consultar en: users.ugent.be/~nverbies

Test acc.		Kappa		Red.		Time	
SPCAD	**0.7920**	**SPCAD**	**0.5942**	CHC	0.9788	POP	0.0620
SSMA	0.7830	SSMA	0.5756	SSMA	0.9653	CNN	0.2757
RMHC	0.7777	RMHC	0.5686	MCNN	0.9373	MCNN	1.5016
RNG	0.7773	HMNEI	0.5644	GGA	0.9301	FCNN	2.3161
CHC	0.7766	CHC	0.5618	RNN	0.9284	MSS	2.5574
GGA	0.7733	RNG	0.5601	CCIS	0.9213	CCIS	4.1314
MoCS	0.7710	MoCS	0.5575	CPruner	0.9070	MoCS	5.1014
HMNEI	0.7679	GGA	0.5572	RMHC	0.9010	AllKNN	8.1177
AllKNN	0.7662	AllKNN	0.5374	DROP3	0.8452	MENN	12.1973
POP	0.7541	POP	0.5298	ICF	0.7452	**SPCAD**	**27.6379**
MENN	0.7520	MENN	0.5175	IB3	0.7197	ICF	30.4970
RNN	0.7519	MSS	0.5172	FCNN	0.6638	HMNEI	66.5790
MSS	0.7460	RNN	0.5100	CNN	0.6187	IB3	185.2876
FCNN	0.7361	FCNN	0.5073	Reconsistent	0.5886	CPruner	194.9527
IB3	0.7347	IB3	0.5041	HMNEI	0.5464	Reconsistent	534.1992
CNN	0.7336	CNN	0.5026	MSS	0.4750	RNG	616.2494
DROP3	0.7139	DROP3	0.4686	MENN	0.4507	SSMA	2084.4178
Reconsistent	0.7083	Reconsistent	0.4659	**SPCAD**	**0.3557**	CHC	2244.7731
CPruner	0.6971	MCNN	0.4407	AllKNN	0.3180	DROP3	2296.5474
CCIS	0.6820	ICF	0.4118	RNG	0.2081	RMHC	3962.0273
MCNN	0.6819	CCIS	0.4016	MoCS	0.1113	GGA	6965.3978
ICF	0.6773	CPruner	0.3894	POP	0.0767	RNN	8030.0853

Cuadro 1: Resultados medios de SPCAD y de los algoritmos del estado del arte de SP

Todos los métodos salvo RNG son significativamente peores que SPCAD con respeto a la precisión en test. Con respeto a kappa, SPCAD claramente supera a todos los demás algoritmos SP considerados.

La tasa de reducción de SPCAD es similar a la de los demás métodos de edición. El tiempo de ejecución de SPCAD es aceptable: SPCAD es mucho más rápido que los demás métodos de envoltura, pero más lento que algunos de los filtros. Esto es interesante: SPCAD es un método de envoltura, pero su naturaleza decremental permite una implementación eficaz, lo que resulta en un tiempo de ejecución bastante bajo.

4. Conclusión y Trabajo Futuro

En este artículo hemos presentado un nuevo método de selección de prototipos, SPCAD. Este método de preprocesamiento elimina instancias ruidosas e intenta mejorar el rendimiento del clasificador *k*NN. Hemos comparado SPCAD con 21 algoritmos del estado del arte de SP para 58 conjuntos de datos. Los resultados obtenidos muestran un rendimiento excelente: SPCAD supera a todos los algoritmos significativamente con respeto a precisión y kappa.

Para mejorar la tasa de reducción de SPCAD, planeamos combinarlo con métodos de condensación para desarrollar un algoritmo híbrido de selección de prototipos basado en conjuntos aproximados difusos. Además, dado que *k*NN es susceptible a los datos de alta dimensionalidad, planeamos abordar dicho problema combinando SPCAD con selección de características.

Referencias

[1] David W. Aha and Dennis Kibler. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[2] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[3] R. Barandela, F. J. Ferri, and J. S. Sanchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intellgence*, 19:787–806, 2005.

[4] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *7th International Conference of Database theory (ICDT)*, pages 217–235, 1999.

[5] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6:153–172, 2002.

[6] Carla E. Brodley. Recursive automatic bias selection for classifier construction. *Machine Learning*, 20:63–94, 1995.

[7] J. R. Cano, F. Herrera, and M. Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575, 2003.

[8] Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[9] C. Cornelis, R. Jensen, G. Hurtado, and D. Slezak. Attribute selection with fuzzy decision reducts. *Information Sciences*, 180(2):209–224, 2010.

[10] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[11] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[12] V. Susheela Devi and M. Narasimha Murty. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

[13] D. Dubois and H. Prade. Rough fuzzy sets and fuzzy rough sets. *International Journal of General Systems*, 17:191–209, 1990.

[14] Robert P. W. Duin and Pavel Paclík. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39:189–208, 2006.

[15] S. García, J. Derrac, J.R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *To appear in: IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[16] Salvador García, José Ramón Cano, and Francisco Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41:2693–2709, 2008.

[17] G. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[18] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[19] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 18:515–516, 1968.

[20] Kazuo Hattori and Masahito Takahashi. A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition*, 32:521–528, 2000.

[21] R. Jensen and C. Cornelis. Fuzzy-rough instance selection. In *Proceedings of the 19th International Conference on Fuzzy Systems (FUZZ-IEEE 2010)*, pages 1–7, 2010.

[22] Ludmila I. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16(8):809–814, 1995.

[23] Ludmila I. Kuncheva and Lakhmi C. Jain. Nearest neighbor classifier: Simultaneous editing and feature selection. *Pattern Recognition Letters*, 20:1149–1156, 1999.

[24] M. T. Lozano, J. S. Sanchez, and F. Pla. *Using the geometrical distribution of prototypes for training set condensing*, volume 3040. 2003.

[25] Elena Marchiori. Hit miss networks with applications to instance selection. *Journal of Machine Learning Research*, 9:997–1017, 2008.

[26] Elena Marchiori. Class conditional nearest neighbor for large margin instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:364–370, 2010.

[27] Z. Pawlak. Rough sets. *International Journal of Computer Information Science*, 11:341–356, 1982.

[28] J.C. Riquelme, J.S. Aguilar-Ruiz, Jesus S. Aguilar-ruiz, and M. Toro. Finding representative patterns with ordered projections. *Pattern Recognition*, 36(4):1009–1018, 2003.

[29] J. S. Sánchez, F. Pla, and F. J. Ferri. Prototype selection for the nearest neighbour rule through proximity graphs. *Pattern Recognition Letters*, 18:507–513, 1997.

[30] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 293–301, 1994.

[31] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, 6(6):448–452, 1976.

[32] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, (6):80–83, 1945.

[33] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286, 2000.

[34] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18:183–190, 1988.

[35] L.A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[36] Ke-Ping Zhao, Shui-Geng Zhou, Ji-Hong Guan, and Ao-Ying Zhou. C-pruner: an improved instance pruning algorithm. In *International Conference on Machine Learning and Cybernetics*, volume 1, pages 94–99, 2003.