Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Ciencias de la Computación

Aprendizaje basado en ejemplos mediante el grafo de semi-espacios proximales

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Ariana Talamantes Alvarez

Ensenada, Baja California, México

2021

Tesis defendida por

Ariana Talamantes Alvarez

y aprobada por el siguiente Comité

Dr. Edgar Leonel Chávez González

Director de tesis

Dr. José Alberto Fernández Zepeda

Dra. Sharon Zinah Herzka Llona

Dr. Pedro Gilberto López Mariscal

Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon

Director de Estudios de Posgrado

Resumen de la tesis que presenta Ariana Talamantes Alvarez como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Aprendizaje basado en ejemplos mediante el grafo de semi-espacios proximales

Resumen aprobado por:

Dr. Edgar Leonel Chávez González
Director de tesis

El problema de clasificación en el aprendizaje automático consiste en identificar a cuál, de entre un conjunto de categorías, pertenece una nueva observación. El aprendizaje basado en ejemplos (*Instance-based learning*, en inglés) es un tipo de aprendizaje automático que utiliza los objetos almacenados en una base de datos para predecir la clase de objetos que no se han visto. Su implementación no necesita de entrenamiento y se adapta a los datos que se tienen disponibles, por lo que resulta útil para problemas donde no se tiene una distribución no estacionaria. El filtrado y clasificación de correo es un ejemplo de clasificación donde la distribución de las clases no es estacionaria, ya que co-evolucionan el *spammer* y el detector, con lo que la clasificación cambia con el tiempo. Hay esencialmente un algoritmo de aprendizaje basado en ejemplos y es el algoritmo de k vecinos más cercanos o kNN (k *nearest neighbors*, en inglés) el cual es muy popular por su simplicidad. La principal desventaja de kNN es que cuenta con el hiperparámetro k que es muy difícil de sintonizar. En los últimos años se han propuesto varios métodos para solucionar este problema pero no se ha llegado a un resultado conclusivo en la literatura. En el presente trabajo se presenta el primer algoritmo de aprendizaje basado en ejemplos que no necesita de parámetros a sintonizar. La clasificación se hace mediante el uso del grafo de semi-espacios proximales o HSP (*Half Space Proximal*, en inglés). Su desempeño se compara con el estado del arte y los resultados obtenidos muestran que el clasificador HSP obtiene mayor exactitud de clasificación que kNN, con cualquier k, en una variedad de ejemplos de datos de altas dimensiones, aún cuando se toman en cuenta diferentes reglas de votación. Más aún, la mejora se mantiene incluso cuando se implementa una heurística para aproximar el clasificador HSP y se utiliza un índice probabilístico para acelerar las consultas.

Palabras clave: aprendizaje basado en ejemplos, grafo de semi-espacios proximales, clasificación, regresión

Abstract of the thesis presented by Ariana Talamantes Alvarez as a partial requirement to obtain the Master of Science degree in Computer Science.

Instance-based learning using the half-space proximal graph

Abstract approved by:

Dr. Edgar Leonel Chávez González
Thesis Director

The classification problem in machine learning consists in designating the class that best fits a new observation, given a set of possible classes. Instance-based learning refers to a family of machine learning algorithms predicting the class of new problem instances by comparing them to instances stored in memory. Its implementation does not need training and adapts to available data, therefore it is useful for problems with a non-stationary distribution. Mail filtering and classification is an example of classification where the class distribution is non-stationary, since the spammer and the detector co-evolve, the classification changes over time. There is essentially one instance-based learning algorithm and it is the k Nearest Neighbors rule or kNN, which is quite popular for its simplicity. The main disadvantage of kNN is the hyperparameter k, which is very difficult to tune. In recent years, various methods have been proposed to solve this problem but a conclusive result has not been reached in the literature. The present work presents the first instance-based learning algorithm without tuning parameters. The classification is done through the use of the Half-Space Proximal Graph or HSP. Its performance is compared to the state of the art. The results obtained show the HSP classifier achieves higher classification accuracy than kNN, for any given k, in a variety of high-dimensional datasets, even when applying different voting rules. Furthermore, the improvement sticks even when a heuristic is implemented to approximate the HSP classifier and a probabilistic index is used to speed up queries.

Keywords: instance-based learning, half-space proximal graph, classification, regression

Dedicatoria

A mi familia.

Agradecimientos

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría. No. de becario: 995146.

A mi director de tesis, por su orientación, atención y confianza en mí.

A mis sinodales, por sus observaciones y sugerencias que ayudaron a mejorar mi trabajo de investigación.

A mis compañeros de posgrado, por su amistad y apoyo.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de tablas

Tabla Página

Capítulo 1. Introducción

El *machine learning*, también conocido como aprendizaje automático, aprendizaje de máquina o aprendizaje basado en datos, conforma una de las subáreas más importantes de la inteligencia artificial. Su propósito es crear sistemas que aprendan funciones o mapeos sin la intervención explícita de un experto. Entre las tareas más exitosas están la identificación de patrones, y problemas de regresión y clasificación. Entre su amplio conjunto de aplicaciones prácticas se incluyen los problemas de procesamiento de lenguaje natural, clasificación de texto o documentos, reconocimiento de voz, así como aplicaciones de visión por computadora; entre estas, contamos reconocimiento e identificación de objetos, detección de rostros, recuperación de imágenes basándose en contenido, etcétera.

Los algoritmos basados en redes neuronales comprenden una parte esencial del aprendizaje automático. Su existencia no es reciente; pero en los últimos años cobraron gran popularidad gracias al desarrollo del aprendizaje profundo (*deep learning*). Este desarrollo fue posible principalmente debido a los avances tecnológicos de hardware que permiten mayor capacidad de cómputo y la disponibilidad masiva de datos que existe en la actualidad. Estos componentes son esenciales para que en ciertas aplicaciones los modelos de aprendizaje profundo obtengan los resultados excepcionales que superan a básicamente todo el resto de algoritmos de aprendizaje automático y en muchas ocasiones, a los humanos.

Las redes neuronales profundas han resultado especialmente útiles para aplicaciones de clasificación de imágenes. El problema era considerado dificil de resolver, debido principalmente a la alta dimensionalidad de las imágenes, ya que el número de píxeles es equivalente al número de dimensiones. Para poder utilizar los métodos tradicionales, es necesaria una etapa previa donde se apliquen algoritmos encargados de crear descriptores de las imágenes y, a su vez, algoritmos para reducir la dimensionalidad de los vectores resultantes. Sin embargo, con el aprendizaje profundo se tiene un modelo que, utilizando los datos como ejemplo, puede crear vectores de características cuyas dimensiones se ven reducidas con respecto a los datos de entrada, y posteriormente dichos vectores son clasificados por el mismo modelo.

Debido a su popularidad, comúnmente se piensa que las redes neuronales pro-

fundas se pueden utilizar para resolver cualquier problema; sin embargo, para poder implementarse existen algunas consideraciones que deben tomarse en cuenta. En muchas ocasiones las desventajas son tales que los algoritmos tradicionales de aprendizaje automático resultan más convenientes. El aprendizaje profundo es un modelo matemático que funciona como una caja negra cuya salida no es intuitiva de entender; o dicho de otra forma, su salida no es explicable. Como caso contrario, se tienen los algoritmos de aprendizaje automático explicable como los árboles de decisión, cuya salida es muy intuitiva. Por ejemplo, si se intenta clasificar la imagen de un perro utilizando una red neuronal y ésta entrega como resultado que la imagen corresponde a un gato, sería muy difícil de saber qué fue lo que llevó al modelo a tomar esa decisión. Esto representa un problema, ya que en muchos casos el poder interpretar los resultados es crucial para, por ejemplo, decidir si una persona es o no buena candidata para recibir un préstamo, elegir el tratamiento médico de un paciente, o básicamente cualquier decisión que deba estar respaldada por una explicación.

Otro punto a considerar respecto al aprendizaje profundo es la eficiencia de su implementación con respecto a otros algoritmos. Las consultas utilizando una red neuronal son muy rápidas; sin embargo, el proceso de entrenamiento para lograr hacer esas consultas es muy lento. Recientemente se probó que el entrenamiento de una red neuronal tiene complejidad ER-completo (*Existential theory of the reals complete*, en inglés), lo que significa que es al menos tan difícil como un problema NP-completo (*Non-deterministic polynomial complete*, en inglés) y que incluso podría no estar en la clase de complejidad NP (Abrahamsen *et al.*, 2021), es decir, que no se podría verificar su solución en tiempo polinomial. Los modelos de aprendizaje profundo pueden tardar días o incluso semanas entrenándose aun teniendo recursos computacionales avanzados como lo son las unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU). En contraste, los métodos tradicionales son más rápidos, generalmente con complejidades polinomiales.

Aunado a lo anterior, cuando se está trabajando en un problema nuevo o no se tiene un modelo de referencia, será necesario invertir más tiempo en el diseño del modelo y en el ajuste de hiperparámetros; por ejemplo, el número de capas ocultas entre la capa de entrada y de salida, o el número de nodos en cada capa. Los hiperparámetros son los datos que rigen el proceso de entrenamiento y para llegar a su configuración

óptima es necesario un proceso iterativo. En este proceso los parámetros se ajustan cada vez que la red neuronal se entrena por completo y se observa la exactitud global.

En muchas ocasiones se tiene la capacidad para invertir tiempo y recursos computacionales para lograr las mejoras en exactitud que ofrecen las redes neuronales profundas. No obstante, existen aplicaciones dinámicas en donde la distribución de los datos cambia con el tiempo. En estas condiciones una etapa de entrenamiento no es posible. Un ejemplo es la detección de SPAM en el correo electrónico. Se necesita un modelo que sea robusto en la detección; pero que no confunda los mensajes importantes con correo no deseado. Para esto, el modelo se puede entrenar con los mensajes que se tienen hasta el momento; sin embargo, conforme pasa el tiempo, el *spammer* puede identificar cómo son los mensajes que se clasifican como no *spam* y de esta manera mejorar los ataques. Debido a esto, lo ideal es estar reentrenando el modelo con los nuevos ejemplos que se presentan para así actualizar las predicciones y controlar los ataques adversarios. En el caso de las redes neuronales profundas, el reentrenamiento continuo no es posible. Se tendría entonces que recurrir a otros métodos de aprendizaje automático que no necesiten entrenamiento.

Finalmente, las redes neuronales usualmente requieren miles o incluso miles de millones de datos etiquetados para entrenarse y así lograr precisiones altas. Este no es un problema fácil de superar porque en muchas ocasiones no se cuenta con tantos datos etiquetados y no hay nada factible que se pueda hacer al respecto.

Considerando lo anterior, en muchas aplicaciones se pueden aprovechar las ventajas tanto de las redes neuronales como de los métodos tradicionales. Un método muy popular en aplicaciones multimedia consiste en utiliza redes neuronales profundas preentrenadas para así obtener vectores de características (*deep features*) y después utilizar otro método de aprendizaje automático para realizar la tarea de clasificación. De esta manera, las redes neuronales únicamente se encargan de convertir las imágenes en vectores representativos de éstas, sin necesidad de ser entrenadas con los nuevos datos.

1.1. Antecedentes

Uno de los algoritmos tradicionales más populares de aprendizaje automático para los problemas de clasificación y regresión es «*k* vecinos más cercanos» (kNN) (Wu *et al.*, 2008). Este algoritmo se cataloga como aprendizaje basado en ejemplos, lo cual indica que el aprendizaje se logra comparando directamente la consulta con los datos almacenados. La popularidad de este algoritmo se debe principalmente a su simplicidad y efectividad. Por otro lado, sus principales desventajas son su complejidad computacional, que es lineal; la elección de función de distancia; y la estimación del hiperparámetro *k*, que es muy difícil de sintonizar. La importancia de este algoritmo se ve reflejada en los numerosos esfuerzos que se han realizado durante los últimos años para solucionar sus desventajas.

La forma directa para encontrar los *k* vecinos más cercanos consiste en calcular la distancia de una consulta hacia todos los objetos en la base de datos para después seleccionar los *k* objetos con la menor distancia. Lo anterior tiene una complejidad lineal y garantiza obtener exactamente los vecinos más cercanos. Una complejidad lineal resulta útil para bases de datos pequeñas pero no es escalable a problemas más grandes. En la actualidad, es muy común tener la necesidad de hacer estas tareas en colecciones grandes y que además se componen de datos complejos; por ejemplo, colecciones de datos multimedia que se representan normalmente por vectores de altas dimensiones (Zezula *et al.*, 2005) como las llamadas «*deep-features*» que se obtienen con modelos de aprendizaje profundo. El problema de encontrar rápidamente los *k* vecinos más cercanos en bases de datos relativamente grandes se ha solucionado gracias al desarrollo de algoritmos para la búsqueda aproximada. Estos algoritmos se implementan en una etapa fuera de línea para obtener complejidades sublineales de búsqueda. En particular, los índices probabilísticos basados en grafos han resultado la mejor opción para espacios de altas dimensiones. El índice «*Hierarchical Navigable Small World*» (HNSW), propuesto por Malkov y Yashunin (2020), es uno de los índices basados en grafos más recientes y que además, según el *benchmark* diseñado por Aumuller *et al.* (2018), es uno de los que obtiene los mejores resultados indexando bases de datos de grandes dimensiones.

Por otro lado, en la implementación más simple de kNN se utiliza la distancia Euclidiana y se le asigna el mismo peso a todos los atributos de los objetos. Se ha notado

que esto puede perjudicar el desempeño del algoritmo cuando se tienen atributos irrelevantes, como es el caso de los datos de alta dimensión. Los enfoques para resolver este problema incluyen asignar diferentes pesos a los atributos o eliminar por completo los atributos menos relevantes. Otros métodos, como el propuesto por Biswas *et al.* (2018), se enfocan en asignar diferentes pesos a los vecinos con la idea de que entre más cercano es el vecino, mayor debería ser su contribución para la predicción de la consulta. También, se han utilizado otras métricas como Mahalanobis (Gautheron *et al.*, 2020; Xiang *et al.*, 2008), Euclidiana adaptativa (Wang *et al.*, 2007) o *Value Difference Metric* (VDM) (Li y Li, 2011).

En cuanto a la elección óptima del hiperparámetro k, el algoritmo básico considera un valor fijo para todas las consultas. Comúnmente se utiliza $k = \sqrt{n}$, propuesto por Lall y Sharma (1996), o se emplean técnicas como validación cruzada (Zhu *et al.*, 2011a) para estimar un buen valor para k. Sin embargo, Zhang *et al.* (2017) resaltaron que el elegir un valor fijo para k no toma en cuenta la distribución de los datos y por lo tanto representa una pérdida de exactitud en las predicciones. Las aportaciones más recientes se enfocan entonces en desarrollar algoritmos que consideren distintos valores de k para cada consulta. Entre el trabajo realizado se encuentran técnicas de cómputo evolutivo (Biswas *et al.*, 2018), métodos probabilísticos (Ghosh, 2006) y modelos lineales (Zhang *et al.*, 2017).

En general, la idea detrás de estos métodos es encontrar los valores óptimos de k, encontrar la vecindad de k vecinos y aplicar una regla de mayoría. Los métodos que se basan en este enfoque requieren procesamiento adicional únicamente para encontrar k; por lo tanto, la complejidad total del algoritmo aumenta. De acuerdo con Zhang *et al.* (2018), estas técnicas tienen una complejidad computacional al menos cuadrática durante el tiempo de clasificación, lo cual no es adecuado para colecciones grandes de datos. Debido a esto, Zhang *et al.* (2018) propusieron los métodos *kTree* y *k*Tree*, donde se introduce una etapa fuera de línea para encontrar los valores óptimos de k. Además de cumplir con encontrar k, estas propuestas también se enfocan en reducir la complejidad computacional presente en los métodos anteriormente propuestos. Con esta modificación, en lugar de tener una complejidad cuadrátrica durante la tarea de clasificación o regresión, se obtiene una complejidad $O(\log d + n)$, donde d corresponde a la dimensionalidad de los datos. Sin embargo, se sigue teniendo una etapa

cuadrática fuera de línea. Por lo tanto, todos los métodos que se han diseñado para la estimación de *k* consisten de una etapa adicional costosa y que además reduce la simplicidad de *kNN*. Adicionalmente, la exactitud que se logra con estos métodos está limitada por el algoritmo tradicional con parámetros óptimos.

1.2. Justificación

El uso de diferentes herramientas ha permitido solucionar la mayoría de las desventajas del algoritmo kNN. Los índices probabilísticos logran que la complejidad del algoritmo se reduzca considerablemente y las redes neuronales profundas funcionan como preprocesamiento para crear vectores de características que pueden utilizarse con diferentes funciones de distancia. Sin embargo, un problema que sigue presente es la selección del hiperparámetro *k*. Durante los últimos años se han propuesto diferentes métodos que buscan solucionar este último problema; sin embargo, los resultados no son conclusivos.

1.3. Hipótesis

- Existe una manera de seleccionar la vecindad que es óptima y no necesita hiperparámetros.
- El vecindario seleccionado por el algoritmo del grafo HSP tiene la propiedad de ser similar y diverso a la vez, lo cual brinda una vecindad representativa que resulta útil para aplicaciones de aprendizaje basado en ejemplos.

Se propone el diseño de un algoritmo basado en ejemplos que funcione de manera similar a kNN pero en el cual la selección del vecindario sea natural, es decir, sin la necesidad de un hiperparámetro como *k*. Para que sea una vecindad de calidad, los vecinos deben ser objetos cercanos a la consulta mientras que al mismo tiempo se proporcione diversidad geométrica entre ellos. Dicho de otra forma, se busca eliminar la redundancia dentro de la vecindad para así lograr una mejor representación del vecindario.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar un algoritmo que supere el estado del arte en clasificación y regresión aplicable a situaciones donde la distribución de los datos varía con el tiempo. En particular, construir un algoritmo basado en ejemplos.

1.4.2. Objetivos específicos

- Diseñar un algoritmo de clasificación basado en ejemplos sin hiperparámetros que supere el estado del arte.
- Diseñar una heurística para obtener una muestra representativa y diversa de una consulta.
- Utilizar la regla de mayoría sobre la muestra representativa para la clasificación basada en ejemplos.
- Implementar una heurística para reducir el tiempo de construcción de la muestra representativa.
- Evaluar la exactitud de clasificación y regresión comparada con el estado del arte.

Capítulo 2. Aprendizaje basado en ejemplos

Los algoritmos de aprendizaje basado en ejemplos (*Instance-based Learning* en inglés) son un tipo de aprendizaje supervisado que utiliza los objetos almacenados en una base de datos para compararlos directamente con objetos nuevos. Estos algoritmos también se conocen como *Lazy Learning* porque no construyen un modelo con los datos disponibles, solo los almacenan. Todo el procesamiento se hace hasta que llega una consulta. El único algoritmo conocido que se clasifica como aprendizaje basado en ejemplos es «k vecinos más cercanos» (kNN).

2.1. Clasificación

La implementación básica de kNN para problemas de clasificación consiste en calcular la distancia entre una consulta y cada uno de los objetos en el conjunto de entrenamiento (i.e. base de datos) para así obtener la vecindad conformada por los k objetos más cercanos, y asignar a la consulta la clase más repetida en el vecindario. El algoritmo kNN necesita almacenar todo el conjunto de entrenamiento y para el aprendizaje solo necesita leer los datos almacenados sin necesidad de hacer algún procesamiento sobre ellos. Además, para su implementación no se necesita que los datos sean linealmente separables, ni conocer la distribución de los datos en general. Estas características lo hacen muy sencillo y muy utilizado.

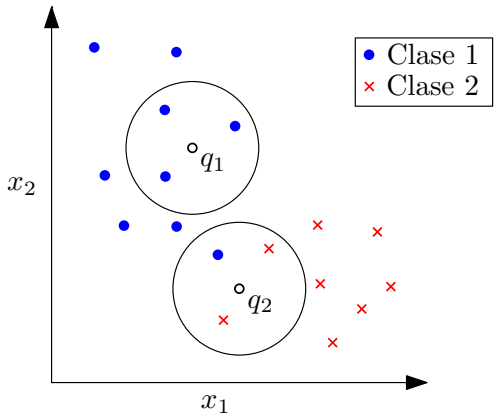


Figura 1. k vecinos más cercanos con $k = 3$ en espacio de dos dimensiones.

El algoritmo de kNN se ilustra en la Figura 1 donde se tiene como ejemplo un valor de $k = 3$ en un problema de clasificación de dos clases con datos de dos dimensiones

(x_1 y x_2). En el ejemplo, la decisión para q_1 se hace directamente porque todos sus vecinos son de la clase 1, por lo que q_1 es también de la clase 1. En cambio, para q_2 se tiene un vecino de la clase 1 y dos vecinos de la clase 2, por lo cual se necesita una regla para decidir qué clase asignar a la consulta q_2. Este caso se puede resolver con una regla de mayoría simple, que le asignaría a q_2 la clase 2, o una regla que agregue pesos a las distancias para dar mayor influencia a los vecinos que estén más cercanos, como se describe en la Sección 2.1.1.

Para utilizar el clasificador kNN es necesario contar con una medida de distancia que permita comparar los ejemplos almacenados con las consultas a clasificar. Dado que en la mayor parte de los casos los datos se representan por vectores en un espacio multidimensional, se supone un conjunto de entrenamiento X conformado por $(x_i)_{i\in[1,n]}$ ejemplos de entrenamiento (donde $n = |D|$). Los vectores de entrenamiento o ejemplos son descritos por un conjunto de F características que equivalen al número de dimensiones. Cada ejemplo está clasificado con una etiqueta $y_j \in Y$. El objetivo es clasificar el ejemplo desconocido q. Para cada $x_i \in X$ se calcula la distancia entre q y x_i como se muestra en la Ecuación 1.

$$d(q, x_i) = \sum_{f \in F} w_f \delta\left(q_f, x_{if}\right) \tag{1}$$

Con esta función se tiene una sumatoria sobre todas las características F con peso w_f para cada característica. Existen muchas métricas que pueden utilizarse; algunas de ellas se describen en la Sección 2.1.2.

Finalmente, además de la función de distancia utilizada para definir la cercanía o similitud entre dos objetos, el desempeño de kNN depende crucialmente del tamaño del vecindario que define el hiperparámetro k. Elegir el valor óptimo de k es una tarea particularmente difícil. Un valor pequeño normalmente resulta en límites de decisión muy sensibles al ruido, mientras que un valor grande produce un vecindario robusto al ruido pero que a su vez es propenso a incluir muchos vecinos de otras clases (véase Figura 2). Usualmente se busca un balance entre ambos casos.

Existe una k óptima para cada distribución de los datos; pero no es posible conocerla, dado que sería necesario saber la etiqueta de la consulta, lo cual resulta tauto-

lógico. Por estos motivos, se han propuesto muchos métodos para encontrar buenos valores para *k*; los cuales, a pesar de ser eficaces, aumentan la complejidad total del algoritmo (Zhang *et al.*, 2018).

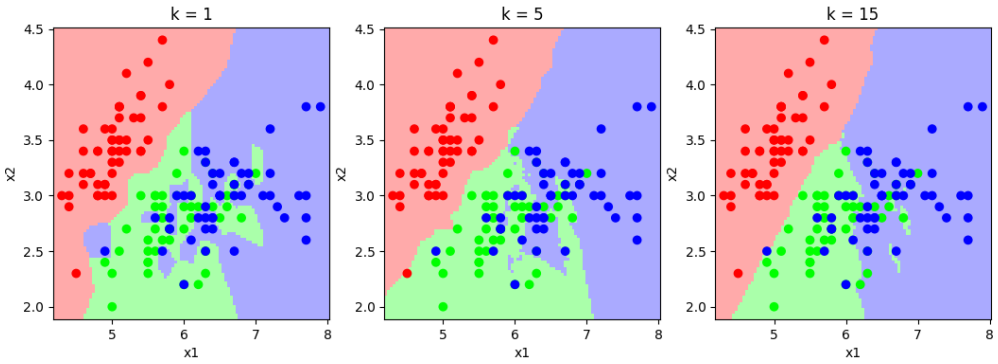


Figura 2. Efecto de la elección de *k*.

La regla de *k* vecinos más cercanos es sólo una manera de obtener una vecindad. Hasta donde conocemos, es la única regla que se utiliza para obtener vecindades en los métodos de aprendizaje basado en ejemplos. Una parte central de este trabajo es una manera alternativa de obtener una vecindad usable en la clasificación basada en ejemplos. La debilidad del clasificador de *k* vecinos es la necesidad de establecer el valor del hiperparámetro *k*, el cual es crítico para obtener buenos resultados en la clasificación. Nuestra tarea será proponer una nueva regla para obtener la vecindad que esté libre de esa limitación.

2.1.1. Reglas de votación

Una vez que se ha utilizado una función de distancia y se han seleccionado los vecinos más cercanos, se necesita una regla para que esos vecinos elijan la clase de la consulta. Para el problema de clasificación, el algoritmo básico de kNN utiliza una regla de mayoría simple, lo que significa que se le asigna a la consulta la clase más repetida entre los vecinos; sin embargo, existen diferentes reglas que se pueden utilizar para hacer la votación. Existen métodos que le asignan mayor influencia a aquellos vecinos que se encuentran más cercanos a la consulta, y una manera sencilla y directa de hacer esto es asignarle a cada vecino un peso correspondiente al inverso de su distancia, como se muestra en la Ecuación 2.

$$w_j = \frac{1}{d\left(q, x_j\right)^p} \tag{2}$$

donde w_j corresponde al peso del vecino x_j y d la distancia entre x_j y la consulta q. El parámetro p normalmente se iguala a 1 pero valores mayores pueden utilizarse para reducir aún más la influencia de los vecinos más distantes. Otro enfoque es utilizar una función exponencial en lugar de la distancia inversa (Cunningham y Delany, 2020), como se ve en la Ecuación 3.

$$w_j = e^{d\left(q, x_j\right)} \tag{3}$$

Por otra parte, se tiene la regla *Distance-Weighted kNN* propuesta por Dudani (1976), definida en la Ecuación 4.

$$w_j = \begin{cases} \frac{d(q,x_k) - d\left(q,x_j\right)}{d(q,x_k) - d(q,x_1)} & d\left(q, x_k\right) \neq d\left(q, x_1\right) \\ 1 & d\left(q, x_k\right) = d\left(q, x_1\right) \end{cases} \tag{4}$$

donde x_k corresponde al vecino más alejado de la consulta q y x_1 al vecino más cercano. En general, estas reglas han demostrado mejor desempeño que la regla de mayoría simple, ya que reducen en cierto grado la sensibilidad del algoritmo a la elección de k (Wilson y Martinez, 2000).

2.1.2. Funciones de distancia

Las funciones de distancia y similitud se utilizan para comparar objetos en una colección. En el aprendizaje basado en ejemplos, estas funciones son muy importantes porque se utilizan para decidir qué objetos almacenados son los vecinos más cercanos a una consulta (cuanto menor es la distancia entre dos objetos, más próximos o más similares son). Una función de distancia debe ser positiva (Expresión 5), simétrica (Ecuación 6), reflexiva (Ecuación 7) y debe cumplir con la desigualdad del triángulo (Desigualdad 8).

$$\forall x, y \in S,\ d(x, y) \geq 0 \tag{5}$$

$$\forall x, y \in S,\ d(x, y) = d(y, x) \tag{6}$$

$$\forall x \in S,\ d(x, x) = 0 \tag{7}$$

$$\forall x, y, z \in S,\ d(x, y) \leq d(x, z) + d(z, y) \tag{8}$$

donde x, y y z son objetos que pertenecen a un conjunto X. La distancia con la que tenemos más familiaridad es la distancia Euclidiana, cuando $X \equiv \mathbb{R}^n$ y se define en la Ecuación 9.

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{9}$$

donde p y q son los objetos a comparar, n es el número de atributos, y p_i y q_i son los valores de los objetos para el atributo i. La distancia Euclidiana es apropiada cuando se tienen atributos numéricos y que además estos atributos se encuentran dentro del mismo rango. Es muy común que esta última propiedad no se cumpla, por lo que usualmente se lleva a cabo una etapa de normalización, en la cual cada atributo de cada objeto es dividido entre la desviación estándar del atributo considerando todos los objetos.

Además de la distancia Euclidiana, existe una gran variedad de funciones de distancia que pueden utilizarse para datos continuos; entre estas funciones se encuentran Mahalanobis (De Maesschalck *et al.*, 2000), Camberra (Lance y Williams, 1966) y Minkowski, que es la forma generalizada de las distancias Manhattan, Euclidiana y Chebyshev. Por otro lado, cuando se tienen datos discretos o categóricos se necesita utilizar una función de distancia que se adapte a estas características. Un ejemplo es *Value Difference Metric* (VDM) (Wilson y Martinez, 2000), la cual se define en la Ecuación 10, considerando dos valores x e y con un único atributo a.

$$VDM_a(x, y) = \sqrt{\sum_{a=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2} \tag{10}$$

donde $N_{a,x}$ es el número de veces que el atributo a tuvo el valor x; $N_{a,x,c}$ es el número de veces que el atributo a tuvo el valor x y la clase fue c; y C es el número de clases. Entre muchos otros ejemplos, se tienen también medidas de similitud como *Context-Similarity Measure* (Biberman, 1994) para valores discretos, que sugiere que la similitud entre dos objetos depende del contexto.

Para casos donde se tiene una combinación de atributos continuos y discretos se puede utilizar *Heterogeneous Value Difference Metric* (HDVM) (Wilson y Martinez, 1997), la cual es una función de distancia que considera varios casos y que contiene la distancia VDM. Esta función no necesita que los datos se normalicen y además funciona para casos donde se tienen valores desconocidos. Esta función se define en la Ecuación 11.

$$HVDM(x,y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{11}$$

donde la función $d_a(x,y)$ es la distancia para el atributo a y se define como se muestra en la Ecuación 12.

$$d_a(x,y) = \begin{cases} 1 & \text{si } x \text{ o } y \text{ son desconocidos} \\ VDM_a(x,y) & \text{si } a \text{ es discreto} \\ \frac{|x-y|}{4_{\sigma_a}} & \text{si } a \text{ es numérico} \end{cases} \tag{12}$$

donde $VDM_a(x,y)$ es la función dada en (10) y σ_a es la desviación estándar de los valores para el atributo a considerando todas las muestras almacenadas. Los valores desconocidos se manejan asignándoles una distancia grande para que las muestras con atributos faltantes tengan menos probabilidades de ser utilizadas que aquellas muestras con todos los atributos especificados.

2.1.3. Elección de k

En el estado del arte se encuentran los métodos *kTree* y *k*Tree* propuestos por Zhang *et al.* (2018). En dicho trabajo se agrega una etapa previa de entrenamiento al

algoritmo kNN para encontrar los valores óptimos de *k* para cada una de las diferentes consultas. En la etapa de entrenamiento se encuentran los valores óptimos de *k* para cada objeto en la base de datos a través de un proceso de reconstrucción y con esta información se construye un árbol de decisión llamado *kTree*. En la etapa de clasificación, el árbol *kTree* se utiliza para encontrar el valor óptimo de *k* para cada consulta y después se lleva a cabo el algoritmo tradicional de kNN utilizando el valor encontrado. El procedimiento se ilustra en la Figura 3.

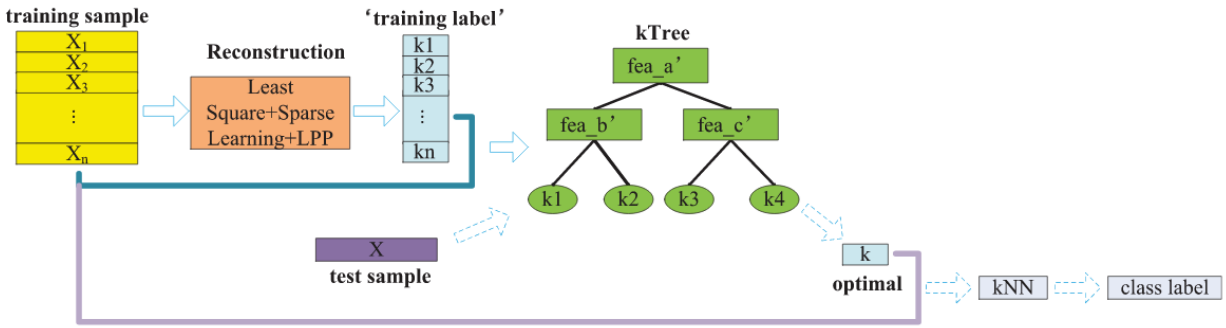


Figura 3. Método *kTree* propuesto por Zhang *et al.* (2018).

La etapa de reconstrucción, encargada de encontrar los valores óptimos de *k*, se basa en la contribución previa de Zhang *et al.* (2014) donde se presenta un modelo para que las muestras de entrenamiento se reconstruyan a sí mismas. El conjunto de entrenamiento se denota como $X \in \mathbb{R}^{nxd} = [x_1, ..., x_n]$, donde n y d representan el número de muestras de entrenamiento y características respectivamente. $W = [w_i, ..., w_n] \in \mathbb{R}^{nxn}$ denota el coeficiente de reconstrucción o las correlaciones entre las muestras de entrenamiento y sí mismas. La función objetivo del proceso de reconstrucción se define como se indica en la Expresión 13.

$$\min_{W} ||XW - X||_F^2 + \rho_1 ||W||_1 + \rho_2 R(W), W \succeq 0. \quad (13)$$

Donde el primer término corresponde a una función de pérdida de mínimos cuadrados con el objetivo de que la distancia entre Xw_i y x_i sea lo más pequeña posible. En el segundo término se utiliza la norma L1 para regularización, con el objetivo de evitar el problema de singularidad y al mismo tiempo obtener una matriz W dispersa.

Finalmente, se agrega un término de regularización con la suposición de que si dos muestras están relacionadas, e.g. x^i y x^j, entonces sus predicciones correspondientes deben tener la misma o similar relación (i.e., $y^i = x^i W$ y $x^i = y^i W$). Este término de regularización se define en la Ecuación 14.

$$R(W) = \mathrm{Tr}(W^T X^T L X W) \tag{14}$$

Donde $L \in \mathbb{R}^{dxd}$ es una matriz laplaciana. Además, en (13) se indica que todo elemento de W es no negativo.

Al optimizar la Función 13 se obtiene la solución W^* óptima, es decir, la matriz de pesos o de correlaciones entre las muestras de entrenamiento. En particular, si $w_{ij} = 0$, se tiene que la i-ésima y j-ésima muestra no están relacionadas. Dicho de otra forma, la muestra i no debería utilizarse para predecir la muestra j. De esta manera, únicamente se utilizan las muestras correlacionadas (aquellas con coeficiente diferente a cero) para predecir cada muestra de entrenamiento. Para ejemplificar, se supone la solución óptima $W^* \in \mathbb{R}^{5x5}$ de la siguiente manera:

$$M = \begin{bmatrix} 0.2 & 0.05 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0.6 & 0.1 \\ 0 & 0.02 & 0.9 & 0 & 0 \\ 0.1 & 0.3 & 0 & 0.8 & 0 \\ 0.02 & 0 & 0.1 & 0 & 0.3 \end{bmatrix}$$

En este ejemplo se tienen cinco muestras de entrenamiento. La primer columna de W^* indica las correlaciones entre la primera muestra y todas las muestras de entrenamiento. Dado que se tienen tres valores diferentes a cero en la primera columna, es decir w_{11}, w_{14}, w_{15}, la primera muestra está relacionada únicamente a dos muestras, excluyendo a sí misma; por lo tanto, el valor óptimo para la primera muestra es $k = 2$. De la misma manera se obtiene el valor óptimo para el resto de las muestras.

Este método se puede utilizar por sí solo para obtener el valor óptimo de k para muestras nuevas y obtener mejoras en las predicciones, a esto se le conoce como

GS-kNN (Zhang *et al.*, 2014). Sin embargo, a pesar de mejorar la exactitud, tiene la desventaja de que el valor de k se obtiene en tiempo cuadrático para cada consulta. Por este motivo, Zhang *et al.* (2018) implementaron un árbol de decisión, llamado *kTree*, compuesto por las muestras de entrenamiento como nodos internos y los valores óptimos de k de cada muestra en las hojas del árbol.

El árbol *kTree* se construye siguiendo la idea del método ID3 (Bahety, 2014) con la diferencia de que el árbol *kTree* considera los valores óptimos de k como las etiquetas de las muestras de entrenamiento. Lo anterior da como resultado que se almacenen diferentes elementos en las hojas del árbol, donde ID3 almacena las etiquetas de las muestras y el *kTree* almacena los valores óptimos de k. Con esta mejora, las muestras nuevas consultan el árbol *kTree*, que se construye en una etapa previa, para obtener su valor óptimo de k y después llevan a cabo el algoritmo tradicional de kNN. De esta manera, la complejidad del algoritmo de kNN con k óptima se reduce a $O(\log d + n)$.

Adicionalmente, el árbol *kTree* se amplía en la versión *k*Tree* donde se agrega información adicional en las hojas del árbol para acelerar las consultas. Esta información incluye para cada muestra de entrenamiento, un subconjunto del conjunto de entrenamiento que contiene a los vecinos más cercanos de cada muestra y los vecinos más cercanos de esos vecinos. De esta manera, las consultas además de obtener el valor óptimo de k, también contienen un subconjunto de muestras sobre el cual llevar a cabo el algoritmo kNN. Con esta modificación se obtiene una complejidad $O(\log d + s)$, donde $s << n$.

2.2. Otras aplicaciones

2.2.1. Regresión

El algoritmo de kNN también se utiliza, menos frecuentemente que para problemas de clasificación, en problemas de regresión, donde lo que se busca es predecir variables continuas (Song *et al.*, 2017). En el caso más simple, la regresión utiliza técnicas estadísticas estándar como la regresión lineal; sin embargo, existen muchos problemas reales que no se pueden predecir con una proyección lineal de valores anteriores. Aplicaciones como la predicción de los precios de las acciones dependen de interacciones complejas de múltiples variables y por lo tanto necesitan técnicas más sofisticadas

para predecir valores futuros. El algoritmo kNN funciona como una herramienta sencilla y efectiva para este tipo de problemas.

Para solucionar problemas de regresión con kNN se sigue el mismo procedimiento que para tareas de clasificación. La diferencia entre ambas tareas se encuentra en el último paso, cuando se decide el valor de la consulta a partir de los k vecinos más cercanos. En lugar de predecir un valor discreto (i.e. la etiqueta de una consulta), se busca predecir un valor continuo. Dado que no se puede aplicar una regla de mayoría, ya que probablemente todos los vecinos tengan valores diferentes, la predicción de la variable continua y de una consulta x normalmente se obtiene con la media (Ecuación 15) o media ponderada (Ecuación 16) del valor de sus vecinos.

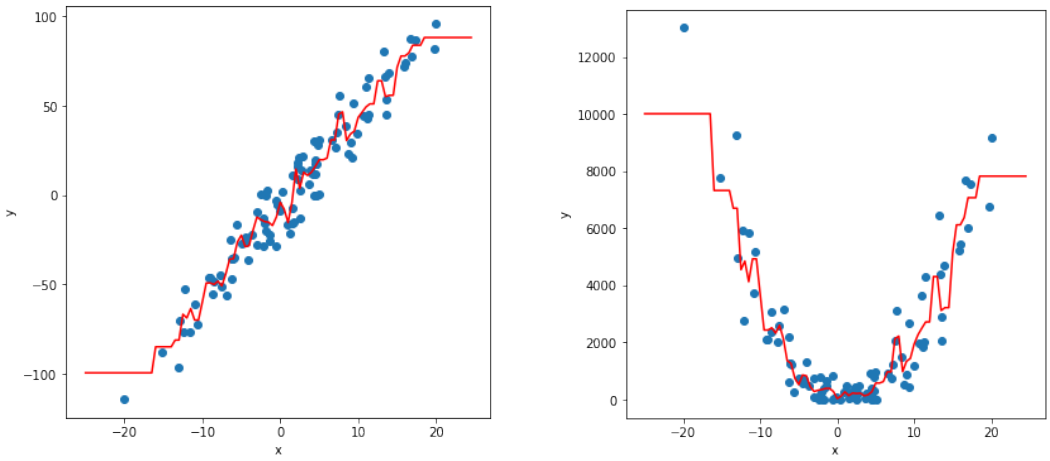
$$y = \frac{1}{k} \sum_{i=1}^{k} y_i(x) \tag{15}$$

$$y = \frac{\sum_{i=1}^{k} y_i(x)\, w_i(x)}{\sum_{i=1}^{k} w_i(x)} \tag{16}$$

Con kNN no se tiene una fórmula cerrada como en otros métodos de regresión sino una predicción para cualquier consulta. Si se consideran todas las posibles consultas y se unen todos los resultados, se obtiene la regresión. Por lo anterior, al igual que en los problemas de clasificación, no es necesario deducir la distribución de los datos, por lo que kNN resulta útil para problemas con diferentes distribuciones (véase Figura 4).

Al igual que en los problemas de clasificación, la elección del valor para el hiperparámetro k representa un problema. La sensibilidad de las predicciones se ve afectada por el valor que se le asigne a este parámetro. En la Figura 5 se tiene un ejemplo de dos dimensiones donde se comparan dos valores diferentes para k. Se puede observar que para un valor pequeño como $k = 1$ se tiene un sobreajuste de los datos. Por otra parte, si se establece un valor grande como $k = 10$, se logra un comportamiento más suave, pero se puede observar que aumenta el error en los bordes, donde no se tienen tantos datos cercanos para elegir a $k = 10$ vecinos de calidad.

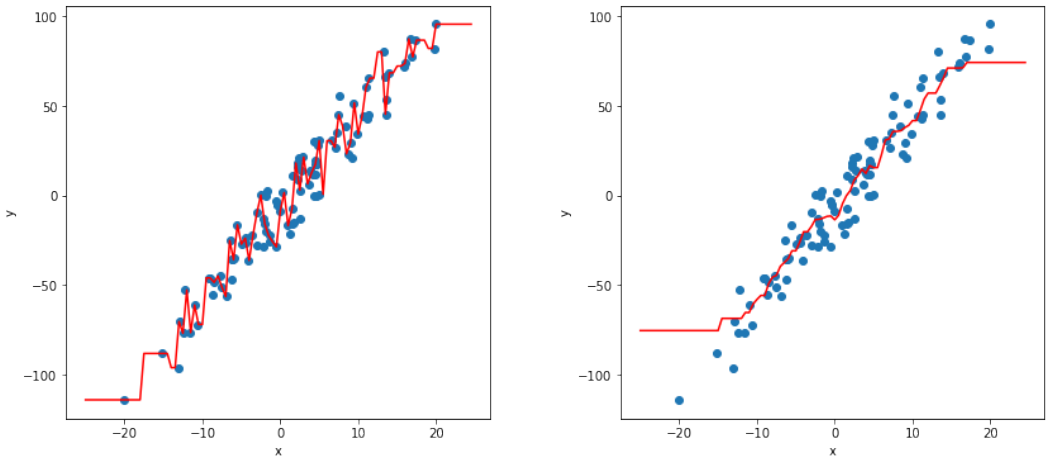
La selección de parámetros tanto en clasificación como en regresión es un problema de optimización; sin embargo, mientras que en la clasificación se busca maximizar



(a) **(b)**

Figura 4. kNN ($k = 3$) para problemas de regresión con diferentes distribuciones.

la exactitud de la predicción de variables discretas, en la regresión se busca minimizar el error de las predicciones para variables continuas. Por ejemplo, si se hace una predicción de un valor numérico como la altura de una persona o el precio de un artículo, el interés no está en saber si el modelo obtuvo exactamente el valor real, sino qué tan cercana estuvo la predicción. Por lo tanto, para evaluar un algoritmo de regresión, se utiliza una función que calcula el error; es decir, la diferencia entre el valor real y el valor estimado.



(a) k=1 **(b)** k=10

Figura 5. Elección de k para problemas de regresión.

Existen muchas métricas para calcular el error y evaluar algoritmos de regresión. Entre las más comunes se encuentran el error cuadrático medio (MSE), ver Ecuación 17, la raíz del error cuadrático medio (RMSE), ver Ecuación 18, y el error absoluto medio (MAE), ver Ecuación 19.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{17}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{18}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{19}$$

donde N es el número de datos, y_i es el valor real esperado y $\hat{y}_i$ es la predicción del modelo.

La elección de error o función de pérdida depende del problema en particular. En ocasiones es particularmente indeseable tener errores grandes, por lo que tiene más sentido utilizar una función que penalice a los puntos que se encuentren más alejados de la media. En estos casos es apropiado utilizar MSE o RMSE, ya que otorgan mayor peso entre mayor sea la diferencia. Por el contrario, en MAE, todas las diferencias individuales tienen el mismo peso. La elección más común es RMSE.

2.2.2. Estimación de densidad

Entre las aplicaciones de kNN también se encuentra la estimación de densidad, la cual es particularmente útil para el análisis exploratorio de datos. Esta tarea consiste en utilizar las observaciones de una muestra aleatoria para estimar la función de densidad de la población general. Cuando una muestra de datos no se parece a una distribución de probabilidad común, como una distribución normal, o no se ajusta fácilmente a una distribución, se dice que tiene una densidad no paramétrica (Fix y Hodges Jr, 1951). En estos casos se utilizan métodos para aproximar la distribución de densidad. Los métodos no paramétricos más conocidos son los histogramas.

El método kNN también se puede utilizar como una herramienta sencilla para la estimación de la densidad no paramétrica; es decir, para cuando los datos no tienen una distribución predefinida. El uso de kNN para esta aplicación la sugirió Loftsgaarden y Quesenberry (1965). El enfoque se basa en el concepto de que el valor de una función de densidad en un punto se puede estimar utilizando el número de observaciones de muestra que se encuentran dentro de una pequeña región alrededor de ese punto. El método kNN estima el valor de densidad en un punto x basándose en la distancia entre x y sus k vecinos más cercanos. Una distancia grande entre x y el k-ésimo vecino indica que la densidad suele ser pequeña y viceversa. Además, las diferentes reglas de votación en el algoritmo kNN permiten utilizar métodos que ponderen la contribución de las observaciones de una muestra de datos en función de su relación o distancia a una muestra determinada para la que se solicita la probabilidad; un ejemplo de esto es el trabajo de Biau *et al.* (2011).

La fórmula general para la estimación de la densidad establece que, para n muestras $x_1, x_2, x_3, ..., x_n$, la densidad en un punto x_0 se puede aproximar con la Expresión 20.

$$\rho(x_0) \approx \frac{k}{nV} \tag{20}$$

donde V es el volumen de alguna vecindad alrededor de x_0 y k denota el número de muestras que están contenidas dentro de la vecindad. Los dos principales enfoques que se utilizan son las ventanas de Parzen y kNN. El primer enfoque elige un valor fijo para el volumen V y determina la k correspondiente a partir de los datos contenidos en la vecindad. Por el contrario, el enfoque de kNN elige un valor fijo para k y determina el volumen V correspondiente a partir de los vecinos seleccionados. En las ventanas de Parzen todas las ventanas tienen el mismo tamaño con diferente número de vecinos cada una, mientras que en kNN el número de vecinos es fijo y las ventanas varían de tamaño. Lo anterior se ilustra en la Figura 6. En condiciones apropiadas y a medida que el número de muestras tiende a infinito, se puede demostrar que ambos métodos convergen al verdadero $\rho(x)$ (Fukunaga y Hostetler, 1973).

El enfoque kNN podría verse como una buena solución para el problema del «mejor» tamaño de ventana, ya que esta varía según k. Sin embargo, como se ha mencionado

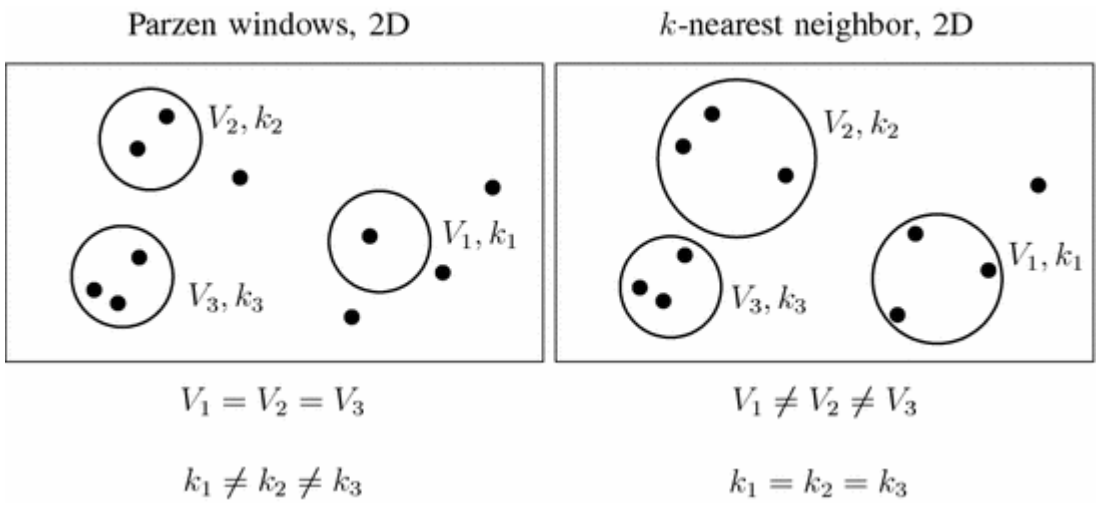


Figura 6. Comparación entre ventanas de Parzen y técnica de *k* vecinos más cercanos (Gramacki, 2018).

anteriormente, con kNN se tiene el problema de la elección de *k*. Entre las contribuciones para solucionar este problema en tareas de estimación de densidad se encuentra el trabajo de Kung *et al.* (2012), donde se propone una distribución asintótica para la elección de *k*. En algunas condiciones, encuentran una estimación de densidad óptima basada en una combinación lineal de vecinos más cercanos.

2.2.3. Imputación de valores perdidos

El aprendizaje automático, y en especial, el aprendizaje basado en ejemplos, depende en gran medida de la calidad de los datos de entrenamiento. En aplicaciones de minería de datos, la tarea que requiere mayor esfuerzo es la preparación de los datos (Qin *et al.*, 2007). Uno de los problemas que se presentan es la existencia de valores perdidos o faltantes, lo cual es muy común en aplicaciones del mundo real. En estos casos, la solución más sencilla es simplemente ignorar las muestras con valores faltantes y únicamente considerar aquellas cuyos datos estén completos. Sin embargo, esta solución tiene el costo de terminar con un conjunto de datos más pequeño y es muy probable que el resultado esté sesgado si los datos faltantes no ocurren en muestras aleatorias.

La imputación es una alternativa a ignorar los datos faltantes. Este método consiste en reemplazar el valor faltante por un valor estimado basado en el resto de datos disponibles. Para esto, se puede utilizar la media, mediana o moda de la variable

considerando el resto de las muestras. Sin embargo, lo más probable es que exista una correlación entre el valor perdido y otras variables, por lo que se podría obtener un mejor resultado haciendo una regresión de la variable con datos faltantes sobre el resto de las variables cuyos datos están completos.

Una solución ampliamente utilizada para este problema de imputación se basa en la técnica de aprendizaje basado en ejemplos (Tang *et al.*, 2020; Zhang *et al.*, 2017). El procedimiento es el mismo que kNN para tareas de clasificación y regresión. Para una variable discreta, kNN usa el valor más frecuente entre los k vecinos más cercanos y, para una variable continua, usa la media o la moda. Los problemas más imporantes que se presentan son la complejidad computacional y la elección del hiperparámetro k.

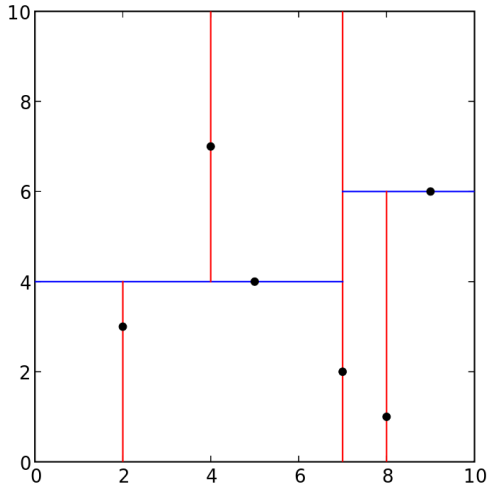
2.3. Complejidad Computacional

La implementación básica de *kNN* utiliza la distancia Euclidiana y tiene una complejidad $O(dn)$ donde n es la cantidad de datos y d la dimensión de estos. La distancia Euclidiana es lineal en el número de características y la cantidad de evaluaciones de distancia crece linealmente con el número de datos. Por estas razones ha habido una cantidad considerable de investigación orientada a reducir el tamaño del conjunto de entrenamiento (Wilson y Martinez, 2000) así como su dimensionalidad (Jiang *et al.*, 2007). Además de esto, se ha hecho mucha investigación para utilizar alternativas a la búsqueda exhaustiva (fuerza bruta) que se utiliza en el algoritmo estándar de kNN, desde métodos exactos hasta métodos aproximados (Li *et al.*, 2020).

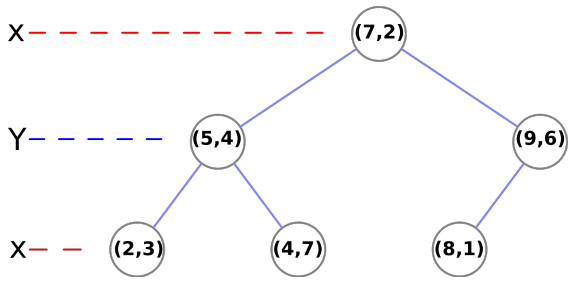
Para el caso de datos en $\mathbb{R}^n$, una estrategia muy utilizada para acelerar la búsqueda exacta de kNN es el *árbol kd* (conocido como *kd-tree* en inglés). La idea general de esta estrategia es utilizar un árbol binario para dividir sucesivamente el conjunto de datos. Las divisiones se realizan utilizando hiperplanos perpendiculares a los ejes; a medida que uno desciende por el árbol, se recorren los ejes utilizados para seleccionar dichos hiperplanos. Los ejes se van alternando en cada nivel del árbol; por ejemplo, en $\mathbb{R}^3$, la raíz tendría un plano perpendicular a x, los hijos de la raíz tendrían planos perpendiculares a y, los hijos de los hijos de la raíz tendrían planos perpendiculares a z iterando las tres dimensiones alternadamente.

Dado que hay muchas formas posibles de elegir hiperplanos para crear semiespa-

cios, existen muchas formas diferentes de construir un *árbol kd*. En el método canónico de construcción del *árbol kd*, los puntos se insertan seleccionando la mediana de sus valores con respecto a sus coordenadas en el eje correspondiente al hiperplano. En la Figura 7 se muestra un ejemplo de cómo se construye el *árbol kd*. En el ejemplo, se comienza por el eje horizontal donde la mediana de los datos es 7, después los datos se dividen con respecto a este valor y se repite el procedimiento continuando con otro eje, que en este caso es el eje vertical. El algoritmo termina cuando las particiones tienen un tamaño mínimo, e.g., tamaño uno.



(a) By KiwiSunset at the English-language Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=16242249



(b) By MYguel - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4535483

Figura 7. Árbol kd. La partición del espacio 2D mostrado en 7a corresponde al árbol en 7b.

Al finalizar se tiene la invariante de que para cualquier nodo, todos los nodos del subárbol izquierdo están de un lado de la división y todos los nodos del subárbol derecho están en el otro lado. Al hacer una consulta, las propiedades del árbol se utilizan para eliminar rápidamente grandes porciones del espacio de búsqueda. El *árbol kd* reduce la complejidad de la búsqueda de $O(dn)$ a $O(d \log(n))$ en promedio; sin embargo, el peor caso sigue siendo lineal. Esta desventaja se debe a lo que se conoce como la *maldición de la dimensión* (Cunningham y Delany, 2020), que hace referencia al crecimiento exponencial del volumen del espacio cuando se aumenta la dimensionalidad, haciendo que los datos se dispersen. Los beneficios de dividir el conjunto de entre-

namiento se acaban cuando se tienen dimensiones grandes. Kleinberg (1997) resalta que cuando $d > \log(n)$, $O(d \log(n))$ no es mejor que $O(dn)$, por lo que resulta mejor opción utilizar fuerza bruta. La búsqueda exacta de k vecinos más cercanos puede superar los problemas de valores grandes de n, pero la dimensionalidad sigue siendo un problema debido a la *maldición de la dimensión*.

Afortunadamente, para la clasificación con kNN y otras aplicaciones no resulta indispensable obtener exactamente los vecinos más cercanos. La búsqueda aproximada resulta útil porque los vecinos que están cercanos (pero no necesariamente los más cercanos) son probablemente de la clase correcta (Cunningham y Delany, 2020). Debido a esto y a las limitantes de la búsqueda exacta, se ha impulsado el desarrollo de índices probabilísticos para la búsqueda aproximada (*ANN*) (Li *et al.*, 2020). La investigación en *ANN* se centra en reducir la complejidad computacional y obtener exactitudes altas. Estos métodos normalmente se usan como una etapa fuera de línea en el algoritmo de kNN para acelerar las consultas.

El método *ANN* de Sunil y Mount (1993) basado en el *árbol kd* permite búsquedas tanto exactas como aproximadas para espacios de varias dimensiones. Este método permite al usuario especificar un límite máximo de error de aproximación, lo que permite controlar el equilibrio entre precisión y tiempo de ejecución. El algoritmo diseñado sirve para dimensiones tan altas como $d = 20$, pero deja de ser útil para dimensiones mayores.

Para colecciones grandes en espacios de altas dimensiones (e.g., $d > 1000$) han sobresalido los algoritmos de *ANN* basados en grafos (Aumuller *et al.*, 2018). Uno de los grafos más eficientes es el grafo *Navigable Small World* (NSW) propuesto por Malkov *et al.* (2014), donde cada inserción encuentra sus vecinos aproximados mediante una búsqueda voraz dentro del grafo parcial construido hasta el momento. El grafo *Hierarchical Navigable Small World* (HNSW) propuesto por Malkov y Yashunin (2020) es una extensión del NSW. Este grafo construye de forma incremental una estructura multicapa que consta de un conjunto jerárquico de grafos de proximidad (capas) para subconjuntos anidados de los elementos almacenados. El grafo HNSW es uno de los índices probabilísticos más eficientes para *ANN* hasta el momento (Li *et al.*, 2020); sin embargo, la investigación en *ANN* es cada vez mayor y continuamente se desarrollan nuevos métodos, los cuales se pueden consultar en *benchmarks* como el desarrollado

por Aumuller *et al.* (2018).

2.4. Estado del arte

El algoritmo kNN fue reconocido como uno de los 10 algoritmos más influyentes en minería de datos por la IEEE en su Conferencia Internacional en Minería de Datos (ICDM) en diciembre del 2006. El objetivo principal de la minería de datos es analizar grandes volúmenes de datos para extraer información relevante que pueda utilizarse posteriormente. Por lo tanto, la minería de datos abarca áreas como aprendizaje automático, estadística y reconocimiento de patrones. Una de las tareas más comunes dentro de la minería de datos es la clasificación.

La selección de los 10 algoritmos más influyentes se hizo a través de una votación en la que participaron los miembros del comité de los programas ICDM ’06, KDD-06 (Conferencia Internacional sobre Descubrimiento de Conocimiento y Minería de Datos, 2006) y SDM ’06 (SIAM Conferencia Internacional en Minería de Datos, 2006), así como los ganadores del premio a la innovación ACM KDD y el premio a las contribuciones a la investigación IEEE ICDM. Además, se consideró por separado la votación de los 145 asistentes de ICDM ’06, obteniendo los mismos resultados que los miembros del comité de los programas mencionados anteriormente (Wu *et al.*, 2008).

La popularidad del algoritmo kNN se debe a que es uno de los algoritmos más sencillos de entender e implementar. Además, a pesar de su simplicidad, ha demostrado obtener exactitudes altas y superar a técnicas más complejas en diferentes aplicaciones. Cover y Hart (1967) demostraron que bajo ciertas suposiciones razonables, el error de kNN está acotado superiormente por el doble del error de Bayes, el cual representa el error de predicción más bajo posible que se puede lograr y es el mismo que el error irreducible. Además, el error de kNN se aproxima asintóticamente al de Bayes y se puede utilizar para aproximarlo.

En la actualidad, kNN se utiliza en aplicaciones como predicción de variables meteorológicas, previsión económica, clasificación de genes, entre otras. Wang *et al.* (2017) realizó un estudio comparativo entre kNN y SVM (*Support Vector Machines*, en inglés) para el pronóstico de energía solar fotovoltaica a corto plazo. Estos métodos se utilizan comúnmente en esta aplicación ya que han demostrado ser una forma eficaz de au-

mentar la precisión de los resultados de la predicción de energía solar, especialmente para la predicción diaria. El principal motivo es que la potencia fotovoltaica depende en gran medida de las condiciones meteorológicas específicas en un periodo de tiempo determinado. Los resultados de la comparación mostraron que SVM funciona bien con una pequeña escala de muestra, mientras que kNN es más sensible al tamaño del conjunto de datos de entrenamiento y puede lograr una mayor precisión que SVM si se tiene el suficiente número de muestras.

Por su parte, Imandoust y Bolandraftar (2013) analizaron el desempeño de kNN para previsión económica. En su estudio demuestran que los métodos no paramétricos como kNN resultan mejor opción que los métodos estadísticos tradicionales. Además, mencionan que la previsión del mercado de valores es una de las aplicaciones más importantes de kNN. Entre las tareas financieras relacionadas con el mercado de valores se encuentra descubrir las tendencias del mercado, planificar estrategias de inversión, identificar el mejor momento para comprar las acciones y qué acciones comprar, entre otras.

En el área de la medicina, el algoritmo kNN se ha utilizado en conjunto con algoritmos genéticos (GA) para encontrar patrones en los niveles de expresión genética que permitan clasificar distintos tipos de tumores (Li *et al.*, 2002). Por otra parte, Pourhomayoun y Shakibi (2021) diseñaron y desarrollaron un modelo predictivo basado en algoritmos de aprendizaje automático para determinar el riesgo para la salud y predecir el riesgo de mortalidad de los pacientes con COVID-19. El modelo desarrollado ayuda a los hospitales e instalaciones médicas a decidir quién necesita atención primero, quién tiene mayor prioridad para ser hospitalizado, clasificar a los pacientes cuando el sistema está saturado y eliminar demoras en la prestación de atención necesaria. Sus resultados mostraron un 93 % de exactitud prediciendo la tasa de mortalidad. Para esto, se utilizaron diferentes algoritmos de aprendizaje automático incluyendo kNN.

En general, el algoritmo kNN se ha implementado con éxito en una variedad de aplicaciones para resolver tareas de minería de datos, como clasificación, regresión, estimación de densidad, e imputación de valores perdidos (Qin *et al.*, 2007). Es una opción popular por su simplicidad y efectividad. Sin embargo, el algoritmo kNN tiene algunos problemas abiertos que deben tenerse en consideración, principalmente, la medición de la similitud entre dos puntos de datos, la complejidad computacional y la

selección del valor k.

La conclusión común con respecto al problema de la medida de similitud es que diferentes aplicaciones necesitan diferentes medidas de distancia (Zhu *et al.*, 2011a; Qin *et al.*, 2007; Zhang *et al.*, 2017). Entre los métodos que se han propuesto se encuentran la similitud coseno, la distancia de Mahalanobis, la distancia de Minkowsky y sus variantes (Simonyan y Zisserman, 2015). Generalmente es posible determinar qué función de similitud es una elección adecuada según las características específicas del problema en particular. Por su parte, la complejidad computacional para la búsqueda de los k vecinos más cercanos se ha logrado reducir con métodos exactos como el *árbol-kd* en dimensiones bajas y métodos aproximados en dimensiones altas, debido al problema conocido como maldición de la dimensión.

Sin embargo, un problema más importante es la selección del hiperparámetro k, es decir, el número de vecinos más cercanos. Este problema ha motivado que se desarrollen métodos complejos para mejorar el algoritmo tradicional kNN. Muchas de estas modificaciones se enfocan en encontrar el valor óptimo de k o reducir en cierto grado el efecto que tiene en la tarea de clasificación o regresión, según sea el caso. Para lo último, puede ser útil ponderar las contribuciones de los vecinos, para que los vecinos más cercanos contribuyan más a la votación que los vecinos más lejanos.

Encontrar el valor óptimo de k es una tarea difícil que se enfrenta al implementar el algoritmo sencillo de *kNN*. La recomendación que se toma como solución sencilla es elegir el valor fijo $k = \sqrt{n}$, donde n es el número de datos. Sin embargo, esto ha demostrado no ser la mejor opción. Además, en muchas ocasiones, quien implementa el algoritmo no está consciente de este problema. La elección de k representa una desventaja importante de kNN.

Capítulo 3. Grafo HSP

Un grafo es una estructura de datos que permite representar interrelaciones entre objetos que interactúan entre ellos. Los grafos son muy utilizados en el área de matemáticas y computación ya que muchos problemas se pueden modelar en forma natural usando grafos; por ejemplo, redes de computadoras, redes sociales, circuitos eléctricos, o estaciones del metro en una ciudad.

Para definir un grafo se usa la relación $G = (V, E)$, donde cada arista $e \in E$ se asocia a un conjunto de nodos o vértices $\{u, v\} \in V$. Si $e \in E$ está asociada a $\{u, v\}$ se dice que u y v son vecinos, o que son adyacentes. Un grafo es no dirigido cuando E es un conjunto de pares no ordenados de elementos de V. Un par no ordenado es un conjunto de la forma $\{a, b\}$, tal que $\{a, b\} = \{b, a\}$. Por el contrario, se dice que un grafo es dirigido cuando E es un conjunto de pares ordenados de elementos de V; es decir, dada una arista (a, b), a es el nodo inicial y b el nodo final.

La cardinalidad de V y E se denota por n y m respectivamente. Se dice que un grafo es denso cuando tiene muchas aristas, usualmente $m = O(n^2)$. Por otro lado, se dice que es disperso cuando tiene pocas aristas, usualmente $m = O(n)$.

Existen distintos tipos de grafos que se definen a partir de sus propiedades y características. En particular, los grafos geométricos son aquellos donde los vértices representan puntos en cualquier posición en el plano y las aristas se trazan como segmentos de línea recta. Los grafos de proximidad son grafos geométricos, construidos sobre un conjunto finito S de puntos en el plano, donde la vecindad de cada vértice se define a partir de cierta noción de proximidad.

El grafo HSP diseñado por Chávez *et al.* (2006) es un grafo de proximidad local que fue propuesto originalmente para aplicaciones en redes ad hoc. Estas redes son un tipo de red inalámbrica descentralizada compuesta por transmisores, llamados *hosts*, que se establece sin necesidad de una infraestructura pre-existente. Computacionalmente, para representar estas redes se utilizan grafos del disco unitario (UDG, por *Unit Disk Graphs* en inglés). Estos grafos están formados a partir de una colección de puntos en el plano euclidiano, en el que dos puntos están conectados si su distancia está por debajo de un umbral fijo.

En la red ad hoc representada como un UDG, los nodos representan los componentes de la red o *hosts* y una arista conecta dos nodos si la distancia euclidiana entre los respectivos *hosts* es menor que una unidad dada, donde la unidad representa el rango común de transmisión de los *hosts* en la red. Es decir, una arista indica que los componentes de la red están en el rango de transmisión.

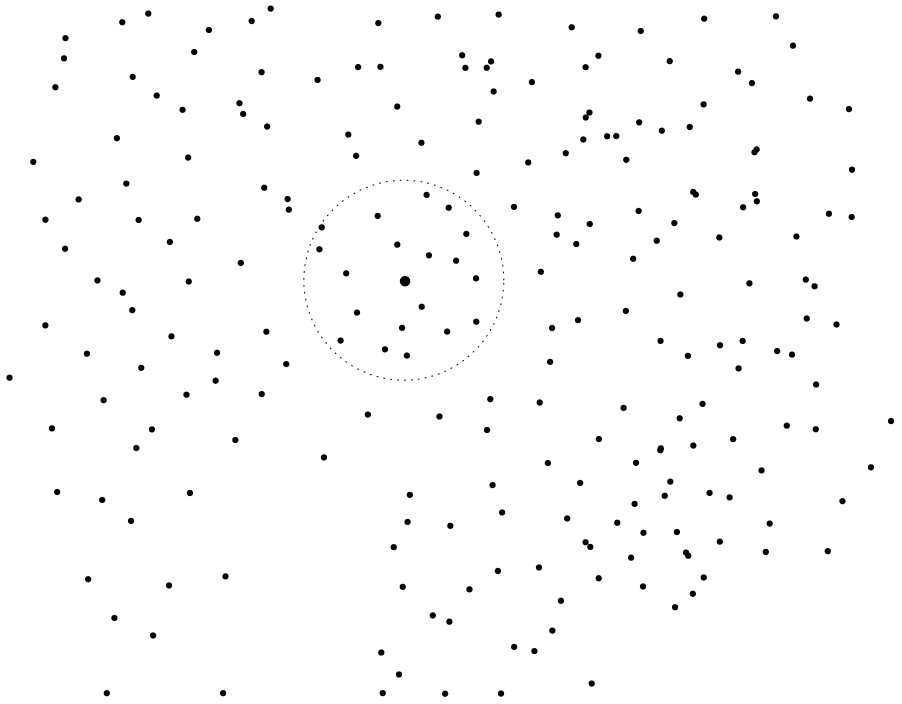


Figura 8. Ejemplo del rango de transmisión de un host. En un UDG, todos los nodos dentro del rango estarían conectados al nodo central.

Es útil extraer un subgrafo UDG para reducir la complejidad de la red; por ejemplo, para optimizar energía en aplicaciones de redes. Además, es deseable que el grafo resultante tenga propiedades como tener un grado pequeño, ser plano (i.e. que ninguna arista se cruce) o tener el árbol de expansión mínimo como subgrafo.

El algoritmo del grafo HSP es una manera de determinar qué vecinos se retienen dentro del rango de cada nodo para construir un subgrafo geométrico adecuado del UDG. El grafo resultante, denominado grafo HSP, es un subgrafo disperso dirigido o no dirigido del UDG. Su construcción es computacionalmente simple y el resultado cuenta con varias propiedades deseables para aplicaciones en redes.

Una característica importante es que utiliza únicamente cálculos locales para su

construcción. Esta propiedad es esencial porque en las redes ad-hoc la topología de toda la red generalmente no está disponible. Además, en redes que cambian dinámicamente, los cambios eventuales deben detectarse y corregirse sin perturbar toda la red.

3.1. Construcción

Para la construcción del grafo HSP, se supone que el grafo $G = (V, E)$ es un UDG donde todo nodo $v \in V$ cuenta con coordenadas (v_x, v_y) en el plano Euclidiano y una etiqueta única (en general un número entero). El algoritmo para elegir los vecinos de cada nodo para construir el grafo HSP se describe en el Algoritmo 1 y se ilustra en la Figura 9.

Algoritmo 1: HSP

Entrada: Un vértice u de un UDG y una lista L_1 de aristas incidentes a u.
Salida: Una lista L_2 de aristas que se retienen de la lista L_1.

1 Establecer el área prohibida $F(u) = \emptyset$;
2 **mientras** L_1 *sea no vacía* **hacer**
3 Remover de L_1 la arista más pequeña, dígase (u, v), (cualquier empate se rompe con el vértice final con la etiqueta más pequeña) e insertar en L_2 la arista dirigida (u, v) con u siendo el vértice inicial;
4 Agregar a $F(u)$ el semiespacio determinado por la línea perpendicular al centro de la arista (u, v) y que contiene al vértice v;
5 Revisar la lista L_1 y remover toda arista cuyo vértice final esté en $F(u)$;
6 **fin**

El área sombreada en la Figura 9 representa los semiespacios que se agregan iterativamente a la región prohibida. Computacionalmente, una arista (u, z) está prohibida por una arista (u, v) cuando la distancia euclidiana de z a v es menor que la distancia euclidiana de z a u. Es importante notar que cada nodo elige a sus vecinos sin necesidad de parámetros. El Algoritmo 1 se lleva a cabo independientemente por cada nodo y el resultado es el grafo HSP que se muestra en la Figura 10.

3.2. Propiedades

El grafo HSP tiene muchas propiedades deseables para redes ad-hoc. Dado un grafo geométrico $G = (V, E)$, un *t-spanner* de G es un subgrafo $G = (V, E')$ tal que para cada

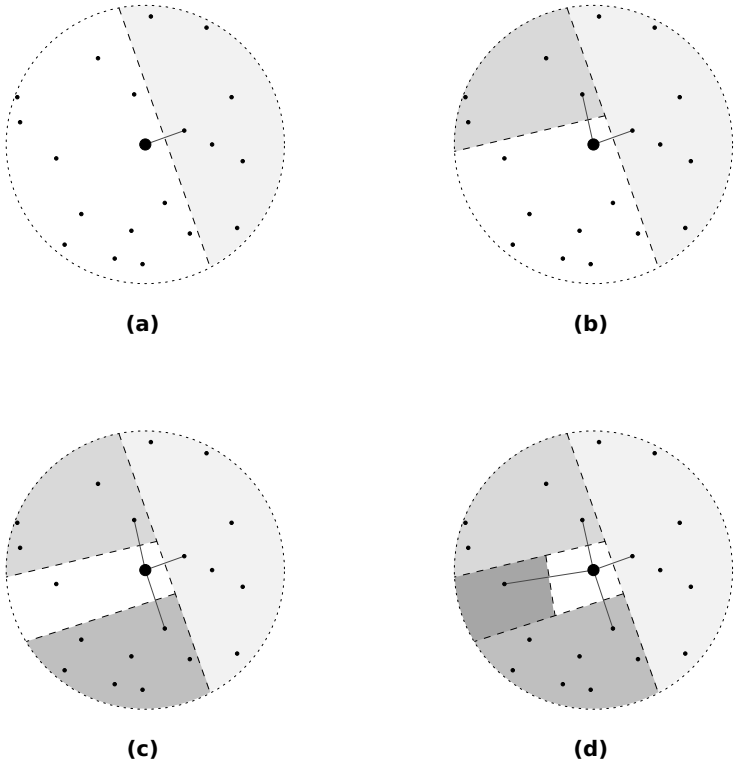


Figura 9. Selección de vecinos para nodo arbitrario.

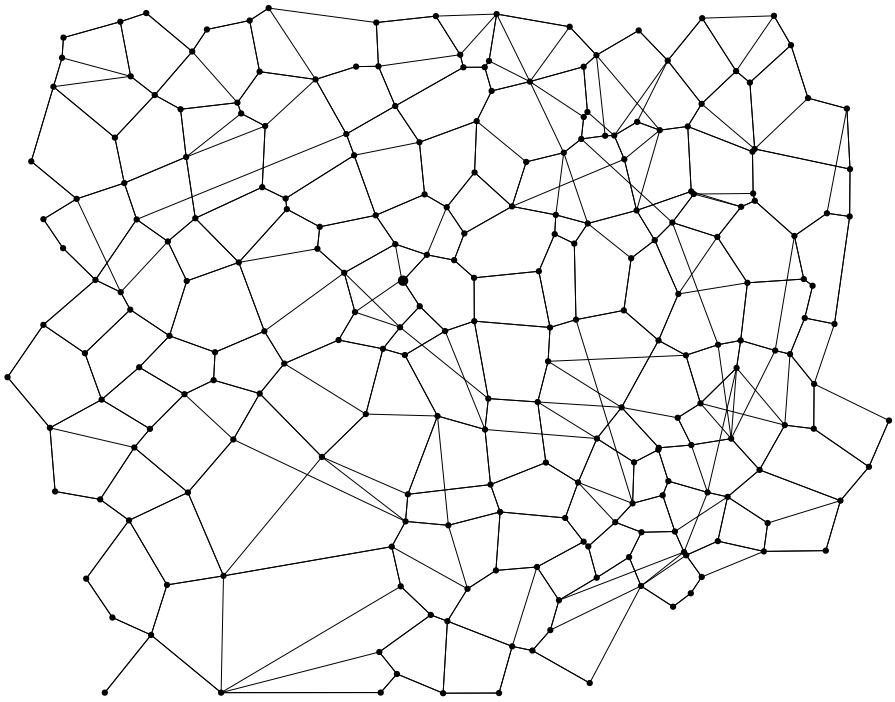


Figura 10. Ejemplo de grafo HSP.

par de puntos $(u, v) \in V$ existe un camino P de u a v cuya longitud es, a lo más, t veces la distancia entre u y v. El grafo HSP es un *t-spanner* con factor de expansión t finito. El *spanner* obtenido es invariante bajo transformaciones de semejanza y contiene el árbol de expansión mínimo. Un árbol de expansión de G es un subgrafo acíclico de G que contiene todos los vértices de G y además es conectado. Sea $w(u, v)$ el peso o costo de la arista conectada entre los vértices $u, v \in V$; un árbol de expansión mínimo es aquel árbol de expansión cuyo peso total sea el menor posible.

El grado de salida del grafo HSP depende de la dimensión intrínseca de los datos y coincide con el número de osculación (*kissing number*, en inglés) en esa dimensión; por ejemplo, es 2, 6, 12 y 24 para las dimensiones 1, 2, 3 y 4, respectivamente. En dimensiones superiores, solo se conocen los límites superior e inferior, con pocas excepciones.

El grafo HSP se puede construir de manera completamente distribuida y es computacionalmente simple. El algoritmo lo ejecuta cada nodo usando solo información de su vecindario. Aunque inicialmente se formuló para datos bidimensionales, donde los vectores representan las coordenadas físicas que corresponden a las ubicaciones geográficas de los componentes en una red, el algoritmo del grafo HSP no tiene un uso explícito de las coordenadas, lo que significa que se puede generalizar para trabajar en cualquier espacio métrico. Por lo anterior, un HSP generalizado se puede utilizar para otras aplicaciones.

Una característica relevante del HSP es que proporciona diversidad y similitud en la vecindad de cada nodo. La vecindad de un nodo comprende a los vecinos después de eliminar la redundancia entre ellos. En particular, lo anterior se puede lograr sin ajustar los parámetros. Esta última propiedad es lo que lo hace único para aplicaciones de aprendizaje basado en ejemplos.

3.3. HSP para aprendizaje basado en ejemplos

En este trabajo se propone utilizar una generalización del grafo HSP para aplicaciones de aprendizaje basado en ejemplos manteniendo la simplicidad del algoritmo kNN. En lugar de comparar los objetos más cercanos a una consulta, se comparan los objetos obtenidos por el algoritmo del grafo HSP y después se lleva a cabo una regla

de votación, como las descritas en la Sección 2.1.1. En esta propuesta se supone que cada consulta puede elegir sus vecinos en toda la base de datos o todo el conjunto de entrenamiento. Otra manera de verlo es que el grafo subyacente no es un UDG, o bien, el radio de transmisión es infinito.

Los vecinos seleccionados tienen la propiedad de ser similares y diversos a la vez, lo que da una vecindad representativa para cada consulta; lo cual es deseable en tareas de aprendizaje basado en ejemplos. Además, tiene la ventaja de ser libre de parámetros, evitando el problema presente en kNN al seleccionar la k. Esta generalización se puede implementar trivialmente en cualquier espacio métrico y cualquier dimensión. El pseudocódigo se presenta en el Algoritmo 2.

Algoritmo 2: Clasificación con HSP

```
   Entrada: conjunto de entrenamiento X y conjunto de prueba Y
   Salida: etiquetas de Y
 1 para cada u ∈ Y hacer
 2     N ← ∅;
 3     C ← X;
 4     mientras C sea no vacío hacer
 5         v ← c ∈ C | d(u, c) ≤ d(u, c'), ∀c' ∈ C;
 6         N.insertar(v);
 7         para cada c ∈ C hacer
 8             si d(c, u) > d(c, v) entonces
 9                 C.remover(c);
10             fin
11         fin
12     fin
13     etiqueta(u) ← etiqueta más repetida en N;
14 fin
```

3.3.1. HSP asintótico

El algoritmo HSP, anteriormente descrito se construye en tiempo lineal; sin embargo, la complejidad del algoritmo HSP no se puede acelerar con un índice probabilístico, como es el caso de kNN. Para solucionar este problema, una alternativa que se propone es calcular el HSP dentro de una bola de cierto radio.

Un radio natural son los k vecinos más cercanos. Calculando el HSP dentro de un área definida por los vecinos más cercanos, las consultas tienen la posibilidad de acelerarse con el uso de un índice, como se describe en la Sección 2.3. Esta modificación

corresponde al HSP asintótico. Además, si se usa un índice probabilístico para calcular los k vecinos más cercanos de la consulta, llamamos a la vecindad resultante HSP asintótico probabilístico.

En el Algoritmo 3 la única modificación se hace en la línea 3, donde los candidatos a vecinos pasan de ser toda la base de datos a únicamente aquellos que se encuentran dentro de los k vecinos más cercanos.

Algoritmo 3: Clasificación con HSP asintótico

```
   Entrada: conjunto de entrenamiento X y conjunto de prueba Y
   Salida: etiquetas de Y
 1 para cada u ∈ Y hacer
 2     N ← ∅;
 3     C ← kNN(X, u, k);
 4     mientras C sea no vacío hacer
 5         v ← c ∈ C | d(u, c) ≤ d(u, c'), ∀c' ∈ C;
 6         N.insertar(v);
 7         para cada c ∈ C hacer
 8             si d(c, u) > d(c, v) entonces
 9                 C.remover(c);
10             fin
11         fin
12     fin
13     etiqueta(u) ← etiqueta más repetida en N;
14 fin
```

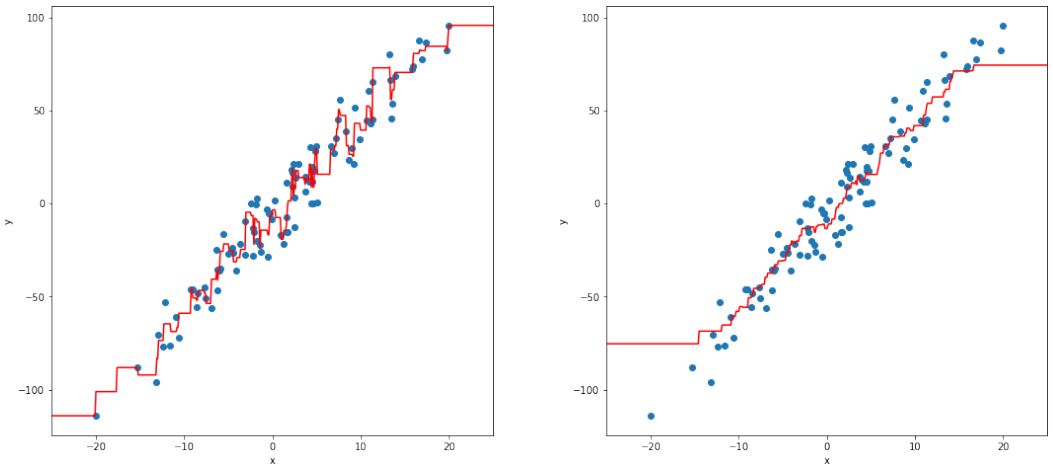
El HSP hace un muestreo del espacio ambiente dando variedad. En cada iteración, después de elegir un vecino, se agregan a un espacio prohibido aquellos objetos que se encuentran más cercanos a ese vecino que a la propia consulta; estos nodos ya no necesitan estar en la vecindad de la consulta porque ya existe un nodo que los represente. Por lo tanto, el hecho de que se defina el espacio prohibido es equivalente a que se nombre un representante por dirección. Con este procedimiento se elimina la redundancia en la vecindad y se obtienen vecinos que son diferentes entre ellos, pero a su vez, cercanos a la consulta. Con este procedimiento se obtiene una buena representación del vecindario.

El grado de un nodo en el grafo HSP está directamente relacionado con la dimensión intrínseca de los datos. La dimensionalidad intrínseca local se refiere a que usualmente los datos se distribuyen en una variedad de dimensiones inferiores cuando solo se

considera un subconjunto cercano. Por lo tanto, un grado alto en el grafo HSP significa que el nodo tiene una dimensión intrínseca alta. Para un trabajo futuro se debe considerar la densidad local de la muestra para darle un grado de confiabilidad a la clasificación. Un grado alto puede significar mayor centralidad de la muestra y por lo tanto, que está mejor representada.

3.3.2. HSP para regresión

Dado que la base del algoritmo HSP para problemas de aprendizaje basado en ejemplos es muy similar a kNN, el algoritmo HSP de igual manera se puede modificar para predecir valores continuos en problemas de regresión. La predicción de la variable continua se puede obtener con la media o media ponderada de los valores de los vecinos, como se describe en la Sección 2.2.1. Al igual que kNN, para implementarse no es necesario conocer la distribución de los datos; el mismo algoritmo se puede implementar para diferentes distribuciones.

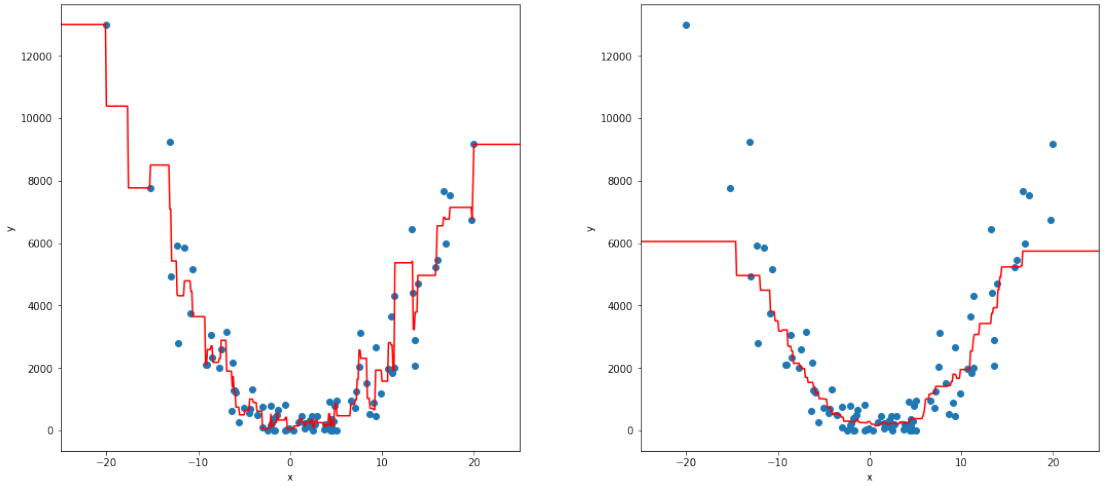


(a) regresión con HSP

(b) regresión con kNN (k=10)

Figura 11. Comparación de regresión con HSP y kNN en distribución lineal.

En el caso de kNN, para cualquiera de sus aplicaciones, la sensibilidad de las predicciones dependen del valor que se le asigne al hiperparámetro k. Los valores pequeños sobreajustan la regresión mientras que los valores grandes pierden precisión cuando no se tienen los suficientes datos para elegir los vecinos. Es por esto que cuando se



(a) regresión con HSP

(b) regresión con kNN (k=10)

Figura 12. Comparación de regresión con HSP y kNN en distribución cuadrática.

implementa el algoritmo kNN se busca encontrar el valor de k que minimice el error. En cambio, la regresión con el algoritmo HSP se adapta a los datos que se tienen disponibles alrededor. Por la naturaleza del algoritmo, se obtiene una regresión única que no puede modificarse con hiperparámetros, y por lo tanto, se tiene un error fijo.

La comparación del comportamiento de ambos algoritmos con diferentes distribuciones de datos se muestra en las Figuras 11 y 12. El comportamiento de kNN para diferentes k se puede observar en la Sección 2.2.1. Para el caso de kNN, la regresión se puede ajustar al modificar el hiperparámetro k.

Capítulo 4. Metodología

Los algoritmos de aprendizaje automático resuelven en esencia un problema de optimización. El teorema NFL (*No free lunch*, en inglés) o «nada es gratis», establece que todos los algoritmos de optimización funcionan igualmente bien cuando se promedia el desempeño considerando todos los problemas posibles (Wolpert y Macready, 1997). Es decir, el teorema implica que no existe mejor algoritmo de optimización. Debido a la relación entre optimización y aprendizaje automático, también implica que no existe el mejor algoritmo de aprendizaje automático para problemas de aprendizaje como la clasificación y la regresión.

En la práctica, el interés no está en todos los posibles problemas de aprendizaje, sino en un pequeño subconjunto de problemas que pueden tener una estructura o forma específica y los algoritmos adaptados a esos problemas. Es por esto que surge la necesidad de desarrollar muchos tipos diferentes de modelos para cubrir la amplia variedad de datos que ocurren en el mundo real y, para cada modelo, contar con una variedad de algoritmos que hagan diferentes compensaciones entre velocidad, precisión y complejidad (Murphy, 2012).

Dada la imposibilidad de modelar con precisión todos los posibles conjuntos de datos y predecir el desempeño de diferentes heurísticas en cualquier conjunto, en problemas de regresión y clasificación se tiene que recurrir a los ejercicios de resultados comparativos ante la ausencia de un modelo universal. Esto no es privativo de la disciplina; en otras áreas, como la de compresión de datos, se realiza la misma práctica. Para saber qué tan bueno es un método de compresión sin pérdidas, es necesario comparar su desempeño con el de otros métodos en diferentes escenarios. Uno de los *benchmarks* que se han desarrollado para este problema es *Pizza&Chilli Corpus* (Ferragina y Navarro, 2005), que contiene diversos archivos de texto lo suficientemente grandes para realizar tareas de compresión y que los resultados sean relevantes y aparentes. Los textos se encuentran disponibles gratuitamente en la web y cubren un conjunto representativo de áreas de aplicación donde el problema de compresión de texto podría ser útil.

De la misma manera, para evaluar problemas de aprendizaje automático como clasificación y regresión, se utilizan bases de datos que permiten comparar el desempeño

de los algoritmos en términos de variables como velocidad y exactitud, dependiendo del enfoque del trabajo. En el presente trabajo de tesis, se realizaron tareas de clasificación y regresión haciendo comparaciones con kNN en diferentes colecciones de datos, y se evaluó la exactitud de los resultados. Se utilizó kNN porque es la única técnica conocida que se puede clasificar como aprendizaje basado en ejemplos. Para hacer los experimentos se seleccionaron conjuntos de datos que se utilizan comúnmente para evaluar comparativamente los nuevos algoritmos que se desarrollan.

Se utilizaron todos los valores necesarios de k para asegurar encontrar el valor que entrega la mayor exactitud posible en kNN. También, se normalizaron los datos y se usaron diferentes reglas de votación con el objetivo de mejorar la exactitud de kNN según lo registrado en la literatura y hacer una comparación justa con nuestro trabajo.

4.1. Conjuntos de datos de referencia

4.1.1. Tareas de clasificación

Se seleccionaron colecciones de datos populares para tareas de clasificación de imágenes. Estas colecciones se transformaron aprovechando las ventajas de las redes neuronales profundas, las cuales se pueden utilizar como un método para crear descriptores o vectores de características, y a su vez, reducir la alta dimensionalidad de las imágenes.

Para obtener los vectores de características se usó lo que se conoce como «transferencia de aprendizaje» (*Transfer Learning*, en inglés). Este método se utiliza en el aprendizaje automático para almacenar el conocimiento adquirido al resolver un problema en particular y utilizar el mismo conocimiento para resolver otro problema diferente pero relacionado. El propósito es mejorar la eficiencia al reutilizar la información recolectada de la tarea aprendida previamente (Pan y Yang, 2010).

La intuición detrás de la «transferencia de aprendizaje» para la clasificación de imágenes es que si un modelo se entrena en un conjunto de datos lo suficientemente grande y general, este modelo servirá efectivamente como un modelo genérico para extraer características de cualquier imagen (Oquab *et al.*, 2014). Dado que la extracción de características es el bloque más complejo de las redes neuronales profundas

(ver Figura 13), la «transferencia de aprendizaje» reduce considerable el tiempo de entrenamiento para un conjunto de datos nuevo, así como la cantidad de imágenes necesarias para entrenar.

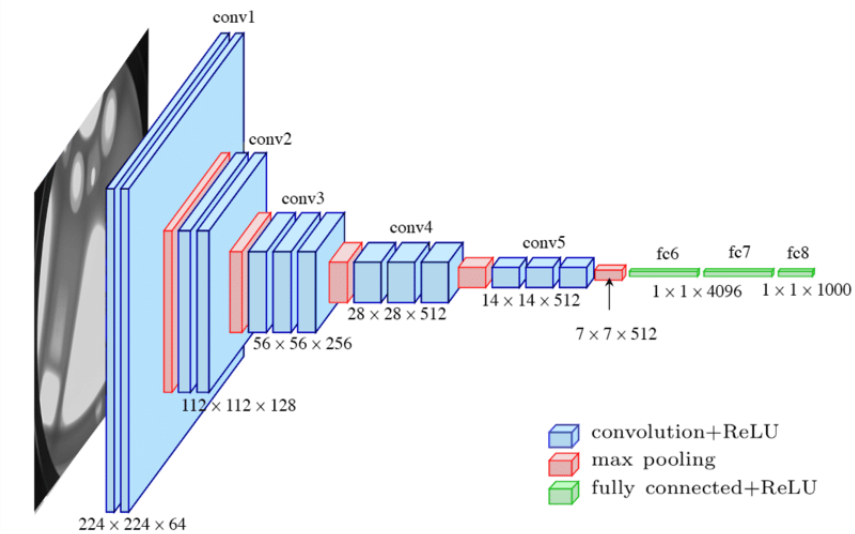


Figura 13. Arquitectura de modelo VGG16 propuesto por Simonyan y Zisserman (2015), donde la etapa de extracción de características comprende de la capa *conv1* a la capa *fc7*.

En este caso se utilizó el modelo VGG16 (Simonyan y Zisserman, 2015) preentrenado con la colección ImageNet, la cual contiene más de un millón de imágenes divididas en 1000 clases diferentes (Deng *et al.*, 2009). Es uno de los conjuntos de datos más utilizados en las últimas investigaciones sobre visión por computadora. Este modelo preentrenado se puede utilizar para clasificar cualquier imagen debido a la cantidad y diversidad de imágenes con las que fue entrenado (Huh *et al.*, 2016).

El modelo VGG16 preentrenado se utilizó únicamente para obtener los vectores de características o «deep features» por lo que se eliminó la etapa de clasificación, que corresponde a la última capa de la red. Se alimentó la red VGG16 con todas las imágenes de nuestro conjunto de datos y se obtuvo el vector de salida de la última capa del bloque de extracción de características. Lo anterior es una técnica ampliamente utilizada para tareas de navegación en colecciones multimedia (Yandex y Lempitsky, 2015; Kratochvíl *et al.*, 2020).

Para la extracción de características se seleccionaron cinco colecciones de datos

con diferentes características; estas colecciones se encuentran entre las más populares para evaluar algoritmos en tareas de clasificación. Las características se describen en la Tabla 1.

Tabla 1. Descripción de colecciones de datos de dimensiones altas.

Nombre	imágenes	características (dimensiones)	clases
MNIST (LeCun y Cortes, 2010)	70000	512	10
Fashion MNIST (Xiao *et al.*, 2017)	70000	512	10
CIFAR10 (Krizhevsky y Hinton, 2009)	60000	512	10
CIFAR100 (Krizhevsky y Hinton, 2009)	60000	512	100
Mini-ImageNet (Vinyals *et al.*, 2016)	60000	2048	100

Los conjuntos de datos tienen una dificultad creciente de clasificación, MNIST, Fashion MNIST, CIFAR 10, Mini- ImageNet y CIFAR 100, respectivamente. Cada conjunto de datos tiene una dimensión que depende del proceso de extracción de características y del tamaño original de las imágenes. Tanto CIFAR10 como CIFAR100 tienen un tamaño original de $(32x32)$, MNIST y Fashion MNIST tienen un tamaño original de $(28x28)$ pero se convirtieron a $(32x32)$ ya que el modelo acepta este tamaño mínimo. Por lo tanto, el tamaño de los vectores de características para estos cuatro conjuntos de datos es el mismo. En el caso de mini-Imagenet, el tamaño de la imagen es $(84x84)$, por lo que la dimensión es mayor. Este conjunto de datos es interesante ya que es un subconjunto de ImageNet. La complejidad de Mini-ImageNet es relativamente alta pero requiere menos recursos que el conjunto de datos completo de ImageNet, cuyas imágenes tienen tamaño $(224x224)$.

4.1.1.1. Evaluación

En cada uno de los conjuntos de datos se seleccionó una muestra aleatoria de 1000 imágenes para utilizar como consultas, con una distribución balanceada de clases. El resto se utilizó como conjunto de entrenamiento o ejemplos. El proceso se repitió 10 veces y se reportó la media y desviación estándar de los resultados. Para evaluar las predicciones correctas se utilizó como métrica la exactitud. La fórmula es la que se muestra en la Ecuación 21.

$$E = \frac{TN + TP}{P + N} \tag{21}$$

donde E es la exactitud que equivale al total de muestras predichas correctamente sobre el total de muestras.

4.1.2. Tareas de regresión

Para las tareas de regresión se utilizaron algunos de los conjuntos de datos públicos del repositorio de datos de aprendizaje automático de la Universidad de California en Irvine (Dua y Graff, 2017), cuyos conjuntos de datos se utilizan para evaluar métodos propuestos para tareas de clasificación y regresión, en términos de precisión y tiempo de ejecución. La base de datos se compone de conjuntos numéricos, categóricos y heterogéneos; también, de conjuntos con valores faltantes y clases desbalanceadas, con el objetivo de evaluar la robustez de los algoritmos. Para este trabajo se seleccionó una variedad de conjuntos numéricos y sin valores faltantes.

La naturaleza de los conjuntos de datos elegidos para las tareas de regresión es distinta a las *deep features* obtenidas con las redes neuronales profundas para las tareas de clasificación de imágenes. En este caso, los vectores representan características que pueden interpretarse individualmente, a diferencia de las *deep features* que son difíciles de explicar o interpretar. Además, el tamaño y dimensionalidad de los conjuntos seleccionados son más pequeños. Las características de estos conjuntos se describen a continuación.

Tabla 2. Conjuntos de datos de repositorio de UCI diseñado por Dua y Graff (2017).

Nombre	muestras	dimensiones
Superconductivity (Hamidieh, 2018)	21263	81
Urban Traffic (Ferreira, 2016)	135	18
Wine Quality (Cortez *et al.*, 2009)	4898	12
Airfoil Self-Noise	1503	6
Appliances energy (Candanedo *et al.*, 2017)	19735	29

Cada conjunto de datos seleccionado corresponde a una aplicación diferente en la que se pueden utilizar métodos de aprendizaje basado en ejemplos para llevar a cabo tareas de regresión. La colección de datos *Superconductivity* (Hamidieh, 2018) consta de materiales superconductores (metales, aleaciones, óxidos, amorfos, etc.) registrados en la literatura y sus propiedades relacionadas. El objetivo de este conjunto

es determinar la temperatura crítica de cada material a partir de sus propiedades superconductoras; por ejemplo, conductividad térmica, coeficiente de Hall y energía termoeléctrica.

La base de datos *Urban Traffic* (Ferreira, 2016) fue creada con registros de comportamiento del tráfico urbano de la ciudad de Sao Paulo en Brasil. Los atributos de cada muestra se conforman por causas típicas de congestión vehicular: autobuses inmovilizados, vehículos en llamas, exceso de vehículos, falta de electricidad, víctimas de accidentes, incidentes relacionados con el transporte de mercancías, fuego, manifestaciones, semáforos apagados e intermitentes, árboles caídos bloqueando el camino, entre otros. El objetivo es estimar la congestión del tráfico a partir de estas variables. El resultado es un número real entre 0 y 100 que representa el porcentaje de congestión en el tráfico.

El objetivo de la base de datos *Wine Quality* (Cortez *et al.*, 2009) es modelar la calidad del vino a partir de pruebas fisicoquímicas hechas a variantes del vino portugués «Vinho Verde»; por ejemplo, porcentaje de alcohol, densidad, acidez fija y pH. La salida es un valor entre 0 y 10 que determina la calidad a partir de esos valores. Es una base de datos interesante porque está desbalanceada, es decir, hay muchos más vinos normales que excelentes o malos, y no aseguran que todas las características o variables de entrada sean relevantes.

Airfoil Self-Noise es un conjunto de datos de la NASA, obtenido a partir de una serie de pruebas aerodinámicas y acústicas de secciones de membranas aerodinámicas bidimensionales y tridimensionales realizadas en un túnel de viento anecoico. El conjunto de datos comprende superficies aerodinámicas NACA 0012 de diferentes tamaños a varias velocidades de túnel de viento y ángulos de ataque. Los atributos incluyen frecuencia, ángulo de ataque, longitud de cuerda, velocidad de flujo libre y espesor de desplazamiento. La salida esperada es el nivel de presión sonora escalado, en decibelios.

Finalmente, la base de datos *Energy* (Candanedo *et al.*, 2017) se utiliza para modelar el uso de energía de electrodomésticos en un edificio de bajo consumo energético. El conjunto de datos se compone por muestras tomadas durante aproximadamente 4.5 meses. Las características incluyen las condiciones de temperatura y humedad en

diferentes habitaciones del edificio. Los datos se obtuvieron a través de una red de sensores inalámbricos ZigBee.

4.1.2.1. Evaluación

Para separar cada colección de datos en el conjunto de prueba y entrenamiento se empleó el método de validación cruzada con diez iteraciones. Específicamente, primero se dividió aleatoriamente todo el conjunto de datos en diez subconjuntos y luego se seleccionó un subconjunto para prueba y los nueve subconjuntos restantes como casos de clase. El proceso se repitió diez veces para evitar el posible sesgo durante la partición del conjunto de datos para la validación cruzada. El resultado final se calculó promediando los resultados de todos los experimentos.

Para la evaluación de resultados se utilizaron el error cuadrático medio (MSE), la raíz del error cuadrático medio (RMSE) y el error absoluto medio (MAE). Se hizo lo anterior con el objetivo de medir con diferentes enfoques la diferencia entre las predicciones y los valores reales. En particular, los errores MSE y RMSE aumentan cuando se tienen valores atípicos o muy alejados de la media, mientras que en el error MAE todas las diferencias tienen el mismo peso.

4.2. Preparación de datos

Antes de poner a prueba los algoritmos, los conjuntos de datos se prepararon reduciendo los parámetros a una escala normalizada. El proceso de normalización consiste en ajustar los valores medidos en diferentes escalas a una escala común. Si los datos no se normalizan, es probable que se encuentran en diferentes escalas, lo que puede causar que algunas características sean dominantes sobre otras características (Trebuňa *et al.*, 2014). El objetivo de la normalización es asignar a cada característica un valor dentro de un rango de 0 a 1, lo cual se puede lograr con muchas técnicas diferentes. Para este trabajo se utilizó la norma Euclidiana o norma L2, la cual se define como la raíz cuadrada de la suma de los cuadrados de los elementos de una matriz, como se muestra en la Ecuación 22.

$$||A||_E = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2} \tag{22}$$

donde a_{ij} es el elemento de la matriz en la i-ésima fila y j-ésima columna, n es el número de filas y m el número de columnas.

4.3. Configuración

Para realizar los experimentos con kNN y nuestro trabajo fue necesario seleccionar una función de distancia y una regla de votación. Dado que los datos son numéricos y el enfoque del trabajo es diseñar un algoritmo de aprendizaje basado en ejemplos que no necesite parámetros, solo se utilizó la distancia euclidiana como función para medir la similitud entre muestras.

Por otra parte, para que los vecinos voten por la clase de la consulta, se realizaron experimentos considerando una regla de mayoría simple. Además, se seleccionaron reglas que consideran la distancia de los vecinos para asignar mayor influencia a aquellos que se encuentran más cercanos a la consulta. Estas últimas se tomaron en cuenta porque han demostrado mejorar el desempeño de kNN (Wilson y Martinez, 2000). Se utilizó la regla propuesta por Dudani (1976) y la técnica general en la cual los vecinos votan sobre la clase de la consulta con votos ponderados por la inversa de su distancia a la consulta. Ambas reglas se describen en la Sección 2.1.1.

4.4. Recursos utilizados

Los experimentos se realizaron dentro de *Google Colab* (Google, 2021). Esta herramienta de Google es un entorno basado en *Jupyter Notebook* que no requiere configuración y permite crear y ejecutar cuadernos interactivos directamente en el navegador. *Jupyter Notebook* (Kluyver *et al.*, 2016) es una aplicación web de código abierto que permite crear y compartir documentos en línea que contienen código ejecutable, ecuaciones, visualizaciones y texto narrativo. Es una herramienta muy utilizada para compartir resultados de investigación. Los usos más populares incluyen: aprendizaje automático, simulación numérica, modelado estadístico y visualización de datos. El

nombre *Jupyter* es una referencia a los tres lenguajes de programación principales soportados por Jupyter: Julia, Python y R.

Google Colab permite usar y compartir cuadernos de *Jupyter* con otros sin tener que descargar, instalar o ejecutar algún programa. Para esto, los cuadernos de *Google Colab* se conectan a una máquina virtual de Google Compute Engine. Esta herramienta permite ejecutar código en la nube de Google y crear modelos de aprendizaje automático con la posibilidad de hacer uso de sus GPU. Además, permite compartir contenido fácilmente y en tiempo real. Esta herramienta se utiliza principalmente para entrenar modelos de aprendizaje profundo, aprovechando el uso gratuito de las GPUs de Google. Los tipos de GPU que están disponibles en *Colab* varían con el tiempo y se asignan a los usuarios según su disponibilidad. Las GPU disponibles en Colab incluyen Nvidia K80s, T4s, P4s y P100s.

Entre las características de la máquina asignada en una versión gratuita, se tienen en promedio 13 GB de RAM y 50 GB de almacenamiento en disco disponibles. Los recursos de *Colab* no están garantizados ni son ilimitados, pero son estables durante el tiempo de vida de la máquina virtual. Los límites de uso varían según la máquina asignada y las necesidades del usuario. En ocasiones, es posible que se asigne automáticamente una máquina virtual con memoria adicional cuando Colab detecta que es probable que se necesite (Google, 2021). El tiempo máximo de conexión a la máquina virtual es en promedio de 12 horas; sin embargo, es posible guardar resultados hasta el momento y reiniciar inmediatamente después la sesión para continuar haciendo uso de la herramienta.

En nuestros experimentos se utiliza la versión predeterminada de *Google Colab*, que es Python 3.7.10. El proceso de *transfer learning* se llevó a cabo usando la biblioteca *Tensorflow* (Abadi *et al.*, 2016) que se utiliza para tareas de aprendizaje automático y está incluida en *Google Colab*. *Tensorflow* es la plataforma más popular para construir y entrenar modelos de aprendizaje profundo.

Se utilizó el modelo predeterminado VGG16 con pesos obtenidos con ImageNet. Esta configuración de modelo es muy utilizada para tareas de transferencia de conocimiento y se encuentra incluída en la biblioteca de *Tensorflow*. Debido a esto no fue necesario entrenar el modelo para obtener los pesos. Se utilizaron las bibliotecas

Sklearn (Pedregosa *et al.*, 2012) y NumPy (Harris *et al.*, 2020) para realizar operaciones con los datos, e.g. validación cruzada, preprocesamiento y evaluación de resultados. Estas bibliotecas se encuentran incluídas en *Google Colab*. Adicionalmente, se instaló la biblioteca *hnswlib* en la máquina virtual de *Google Colab* para hacer uso del índice HNSW (Malkov y Yashunin, 2020) en las versiones probabilísticas de kNN y HSP asintótico.

4.5. Clasificador kNN ideal

En una implementación ideal del clasificador kNN, el algoritmo clasifica correctamente siempre y cuando sea posible. En este escenario, el valor óptimo de k es cualquier valor que permita clasificar correctamente la consulta dada. Dado que el conjunto de valores posibles que clasifican correctamente es diferente para cada muestra, el clasificador ideal kNN utiliza un valor de k diferente en cada consulta. Además, es importante notar que habrá casos en los que el algoritmo kNN clasificará incorrectamente sin importar el valor de k que se elija. En estos casos no existe tal valor óptimo de k.

La exactitud en este caso ideal del clasificador kNN depende entonces de la cantidad de muestras para las que sí existe un valor de k que las clasifique correctamente. Esta exactitud representa la exactitud máxima que podrían alcanzar los clasificadores kNN que utilizan heurísticas para encontrar el mejor valor de k, como *GS-kNN* y *kTree*. Estas heurísticas se desarrollan porque el algoritmo ideal de kNN no existe, ya que sería necesario conocer de antemano la etiqueta de la consulta, lo cual es el propósito de la tarea de clasificación en sí.

Para propósitos de experimentación y contar con una referencia, se hizo un experimento de clasificación para conocer el resultado que tendría un clasificador kNN ideal. Se utilizaron las mismas colecciones de datos que se eligieron para comparar los algoritmos kNN y HSP para tareas de clasificación (Tabla 1). El experimento consistió en clasificar cada consulta del conjunto de prueba utilizando todos los valores de k entre 1 y 300, y contar el número de consultas que se clasificaron correctamente utilizando alguno de esos valores. El propósito de este experimento es conocer el resultado que se obtendría si fuera posible utilizar la k óptima para cada muestra sin saber previa-

mente la etiqueta de esta.

Tabla 3. Exactitud de un clasificador kNN ideal.

Colección	Exactitud
MNIST	95.39 %
Fashion MNIST	94.6 %
CIFAR10	81 %
CIFAR100	55.7 %
Mini-ImageNet	73.3 %

Los resultados muestran el porcentaje de muestras para las que existe al menos un valor de k que las clasifica correctamente. Con este resultado se puede conocer el porcentaje de muestras para las que no sería posible encontrar un valor óptimo de k utilizando alguna de las heurísticas que se han desarrollado, ya que no existe un valor que las clasifique correctamente. Es importante reiterar que esta implementación ideal de kNN no sería posible en un escenario real. Los resultados mostrados en la tabla anterior no son posibles sin saber de antemano la etiqueta de cada muestra. Dado que el propósito de la tarea de clasificación es encontrar la etiqueta de la consulta, si esta se conociera, no habría tarea de clasificación. Sin embargo, este resultado es importante porque proporciona un punto de referencia para evaluar el desempeño de algoritmos como HSP y las variaciones de kNN.

Capítulo 5. Resultados

En varios trabajos recientes (Zhang *et al.*, 2017; Cheng *et al.*, 2014; Zhang *et al.*, 2014) se discute el hecho de que el usar un valor fijo de k en kNN no siempre es razonable en aplicaciones reales, por lo que resulta mejor utilizar un valor diferente de k para cada muestra de prueba. Además, se menciona que los diferentes valores de k deben aprenderse a partir de la distribución de los datos. Sin embargo, las heurísticas que se han desarrollado siguiendo este enfoque resultan muy costosas, especialmente para colecciones grandes de datos.

En los últimos años se han desarrollado varias propuestas para encontrar el valor óptimo de k para cada consulta o muestra en el conjunto de prueba. Entre los métodos que obtienen mayor exactitud se encuentra *GS-kNN* (Zhang *et al.*, 2014). Este método tiene una complejidad cuadrática porque utiliza todas las muestras de entrenamiento para reconstruir cada una de las muestras de prueba y así obtener su mejor valor para k (ver Capítulo 2). Para acelerar las consultas, se tienen los métodos *kTree* y *k*Tree* (Zhang *et al.*, 2017) que utilizan el mismo enfoque que *GS-kNN* pero reducen considerablemente el tiempo de ejecución, logrando clasificar con una complejidad similar a kNN y obteniendo una exactitud de clasificación muy cercana a *GS-kNN*. Sin embargo, los métodos *kTree* y *k*Tree* aún necesitan de una etapa cuadrática. La etapa fuera de línea de los métodos *kTree* y *k*Tree* utiliza el mismo procedimiento que en *GS-kNN*, la diferencia es que solo lo necesitan hacer una vez.

Los algoritmos con complejidades de tiempo cuadrático requieren mucho tiempo para ejecutarse, especialmente cuando se tienen conjuntos grandes de datos en espacios de dimensiones altas. En este escenario se encuentran las aplicaciones de clasificación de imágenes usando descriptores de altas dimensiones como las *deep features*. Por este motivo, los algoritmos como *GS-kNN*, *kTree* y *k*Tree* no son apropiados para los conjuntos de datos seleccionados en este trabajo para llevar a cabo los experimentos de clasificación.

Adicionalmente, es importante notar que para el aprendizaje basado en ejemplos, entre más ejemplos se tengan, es decir, entre mayor sea el tamaño de la base de datos, mejor será el desempeño del algoritmo de clasificación. En contraste, entre mayor sea el tamaño de la base de datos, el tiempo de ejecución de un algoritmo con comple-

jidad cuadrática aumentará hasta que se vuelva imposible llevar a cabo las consultas.

5.1. Clasificación

Para evaluar kNN y compararlo con otros algoritmos, usualmente se selecciona una lista de valores fijos para k en función del número de datos. La elección de valores se basa en trabajos como el de Lall y Sharma (1996) donde proponen $k = \sqrt{n}$. Después, se lleva a cabo el algoritmo kNN utilizando los diferentes valores y se reporta únicamente la mayor exactitud obtenida. Sin embargo, en este trabajo se consideraron todos los valores de k, de 1 a 300, para el clasificador kNN, el HSP asintótico y las versiones probabilísticas. Esto con el objetivo de asegurar obtener la mayor exactitud posible con el clasificador kNN y observar su comportamiento.

Las gráficas muestran la exactitud de la clasificación en el eje vertical. Se puede observar que la exactitud de clasificación de los diferentes clasificadores varía en función de k con excepción del clasificador HSP. La exactitud del clasificador HSP se mantiene constante porque solo hay un resultado, el cual no depende de hiperparámetros como k. Es importante notar que cualquier método para seleccionar una k adecuada para kNN correspondería a un valor entre 1 y 300, prescindiendo de la necesidad de comparar con los clasificadores de kNN en el estado del arte que se enfocan en encontrar la mejor k. Además, se puede observar que el valor óptimo de k es diferente para cada conjunto de datos; es decir, los valores máximos de las series de los distintos clasificadores son distintos. A pesar de ser conjuntos de datos de tamaño y dimensión similar, el elegir un valor fijo de k para todos los conjuntos de datos no sería adecuado.

En la práctica, lo común sería no contar con los recursos para probar todos los valores de k necesarios para encontrar el valor óptimo, y se utilizaría un valor como $k = \sqrt{n}$ o se emplearían técnicas como validación cruzada para estimar un buen valor, pero no necesariamente el mejor. Si lo que se busca es un algoritmo con buen desempeño que se pueda aplicar directamente sin necesidad de optimizar parámetros, kNN no sería la mejor opción. Por otra parte, el clasificador HSP se pone en práctica sin necesidad de modificar hiperparámetros. Este clasificador tiene un buen desempeño porque el vecindario además de brindar similitud con la consulta, también brinda diversidad entre los vecinos. Además, el tamaño del vecindario es diferente para cada muestra y

depende de la dimensión local de los datos.

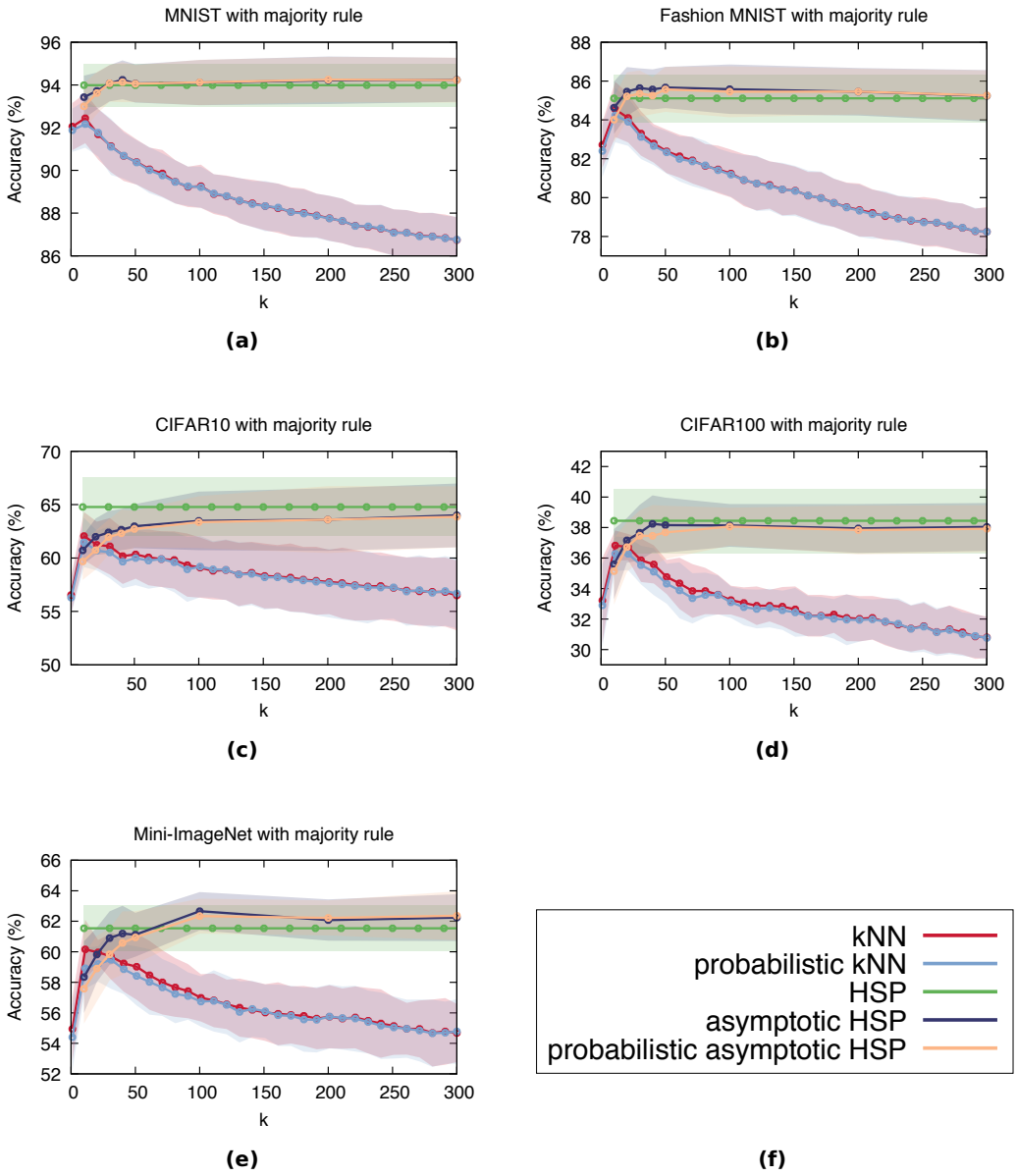


Figura 14. Comparación de clasificadores kNN, HSP y HSP asintótico, utilizando regla de mayoría.

En el primer experimento, mostrado en la Figura 14 se usó la regla de mayoría como regla de votación para los cinco clasificadores. En este caso, la exactitud obtenida con el clasificador HSP y las versiones asintóticas supera a ambas versiones de kNN con cualquier valor de k en todas las bases de datos utilizadas. Además, las versiones asintóticas del HSP tienen un comportamiento más suave que kNN, que parece caótico en función de k. La exactitud del HSP asintótico y su versión probabilística se acercan

a la exactitud del clasificador HSP conforme aumenta el valor de k.

En el segundo experimento, se utilizó la regla de Dudani (1976) como regla de votación. La Figura 15 muestra que, de manera similar al experimento anterior, el HSP y sus versiones asintóticas superan a kNN para todos los conjuntos de datos.

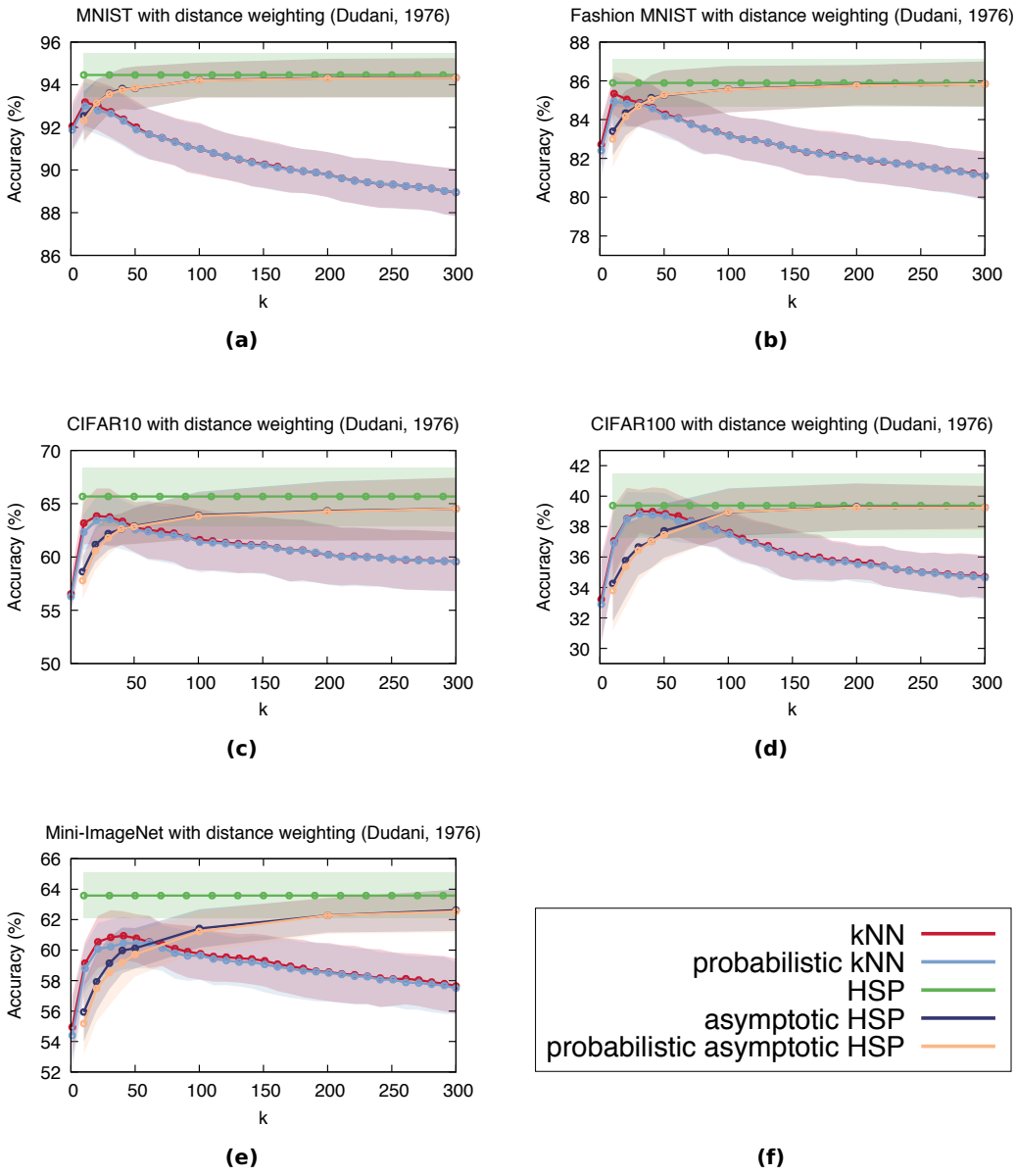


Figura 15. Comparación de clasificadores kNN, HSP y HSP asintótico, utilizando regla propuesta por Dudani (1976).

Se puede observar que los valores óptimos de k en cada caso son diferentes a los obtenidos en el primer experimento, y a su vez, son diferentes entre cada conjunto de

datos. También, se puede observar que el comportamiento de kNN se suaviza ligeramente usando la regla de Dudani que asigna mayor influencia a los vecinos según su distancia, en comparación con la regla de mayoría donde todos los vecinos tienen el mismo peso en la votación. La elección de una regla de votación que agrega diferentes pesos a los vecinos según su distancia resulta benéfico para kNN, ya que reduce en cierta forma la influencia de k.

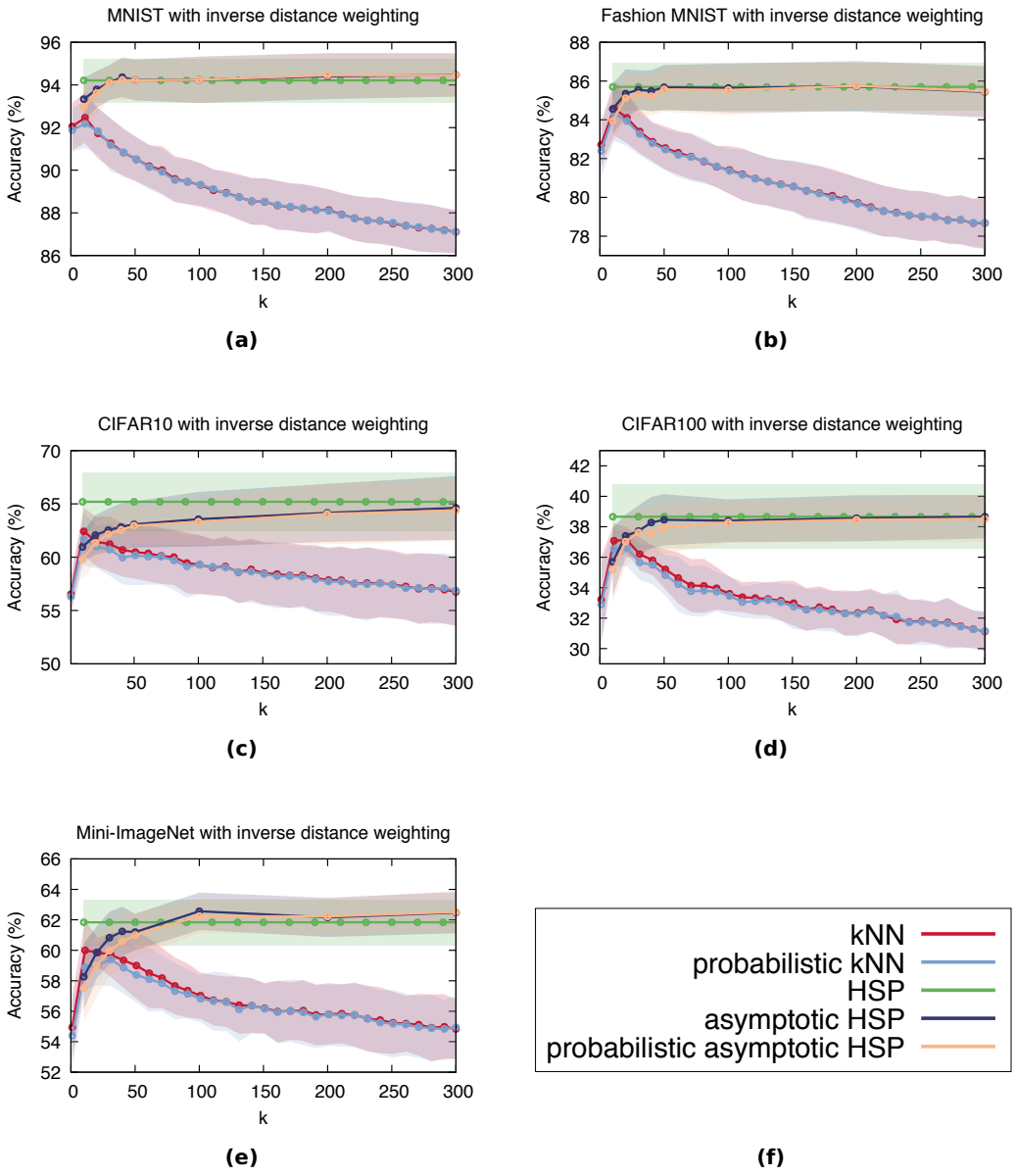


Figura 16. Comparación de clasificadores kNN, HSP y HSP asintótico, asignando a los vecinos pesos correspondientes al inverso de su distancia a la consulta.

Finalmente, en el tercer experimento, para la votación se le asignó a cada vecino un peso correspondiente al inverso de su distancia a la consulta. Se observa en la Figura 16 que el HSP y sus versiones asintóticas superan nuevamente a kNN en todos los conjuntos de datos.

Tabla 4. Porcentaje de exactitud máxima para cada técnica utilizando: 1. Regla de la mayoría, 2. Regla de ponderación de Dudani, 3. Ponderación de distancia inversa.

		CIFAR 10	CIFAR 100	MNIST	Fashion MNIST	Mini ImageNet
1	kNN	62.0	36.8	92.4	84.5	60.1
	P-kNN	61.4	36.2	92.1	84.2	59.4
	HSP	**64.78**	38.44	93.99	85.11	61.54
	A-HSP	64.6	38.5	**94.2**	**85.6**	**62.6**
	PA-HSP	64.6	**38.6**	**94.2**	85.5	62.3
2	kNN	63.8	39.0	93.1	85.3	60.9
	P-kNN	63.5	38.8	93.0	84.9	60.5
	HSP	**65.68**	39.38	94.46	85.89	**63.58**
	A-HSP	65.0	**39.7**	**94.5**	**85.9**	63.4
	PA-HSP	65	**39.7**	**94.5**	**85.9**	63.4
3	kNN	62.4	37.1	92.4	84.5	60.0
	P-kNN	61.7	36.5	92.1	84.3	59.3
	HSP	65.2	38.66	94.21	**85.7**	61.84
	A-HSP	**65.3**	**38.8**	**94.4**	**85.7**	**62.5**
	PA-HSP	65.2	**38.8**	**94.4**	**85.7**	**62.5**

Los resultados se resumen en la Tabla 4 donde destacan la versiones asintóticas del HSP. Es importante resaltar que a pesar de que el HSP asintótico cuenta con un parámetro, este no funciona de la misma manera que en kNN. En el caso del HSP asintótico solamente es necesario elegir un valor de k lo suficientemente grande para que el resultado sea cercano al del HSP, como se muestra en las Figuras 14, 15 y 16. Para ejemplificar esto, la Figura 17 muestra los resultados obtenidos con el HSP asintótico usando el valor fijo de $k = 300$ y la regla de mayoría. Se puede observar que, usando el mismo valor fijo en todos los casos, el HSP asintótico obtiene mayor exactitud de clasificación que kNN para cualquier valor de k.

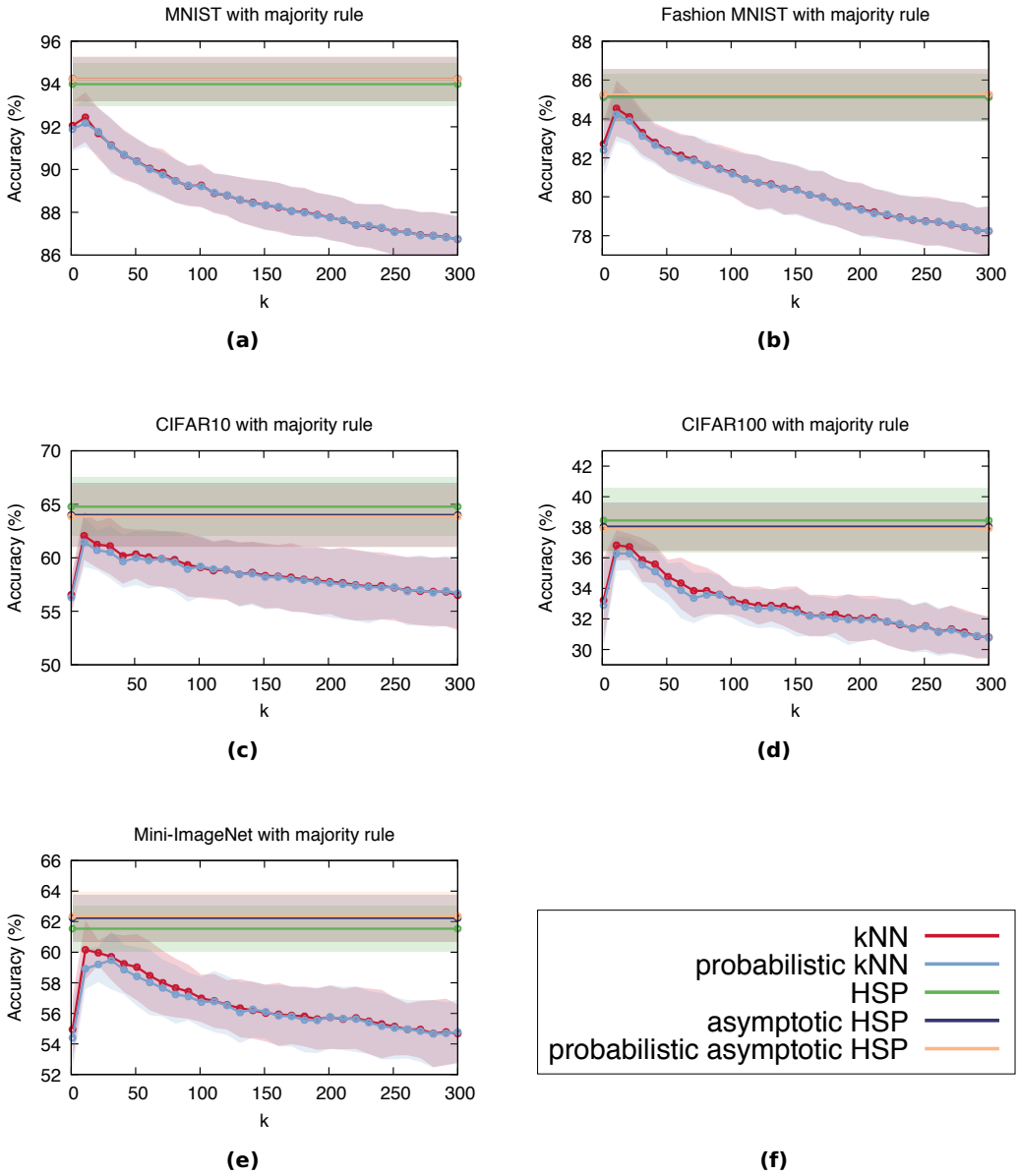


Figura 17. Comparación de clasificadores kNN, HSP y HSP asintótico, utilizando regla de mayoría y valor de k fijo para HSP asintótico.

5.2. Regresión

El desempeño de los algoritmos en problemas de regresión se comparó de manera similar que en las colecciones de datos para problemas de clasificación. Debido a que en este caso el tamaño de las colecciones es menor, se utilizó un menor rango de valores para k. Específicamente, se consideraron todos los valores de k, de 1 a 100, para el clasificacor kNN y el HSP asintótico. A diferencia de los problemas de clasificación,

en los problemas de regresión se tiene un problema de minimización, donde se busca minimizar el error definido por una función de pérdida.

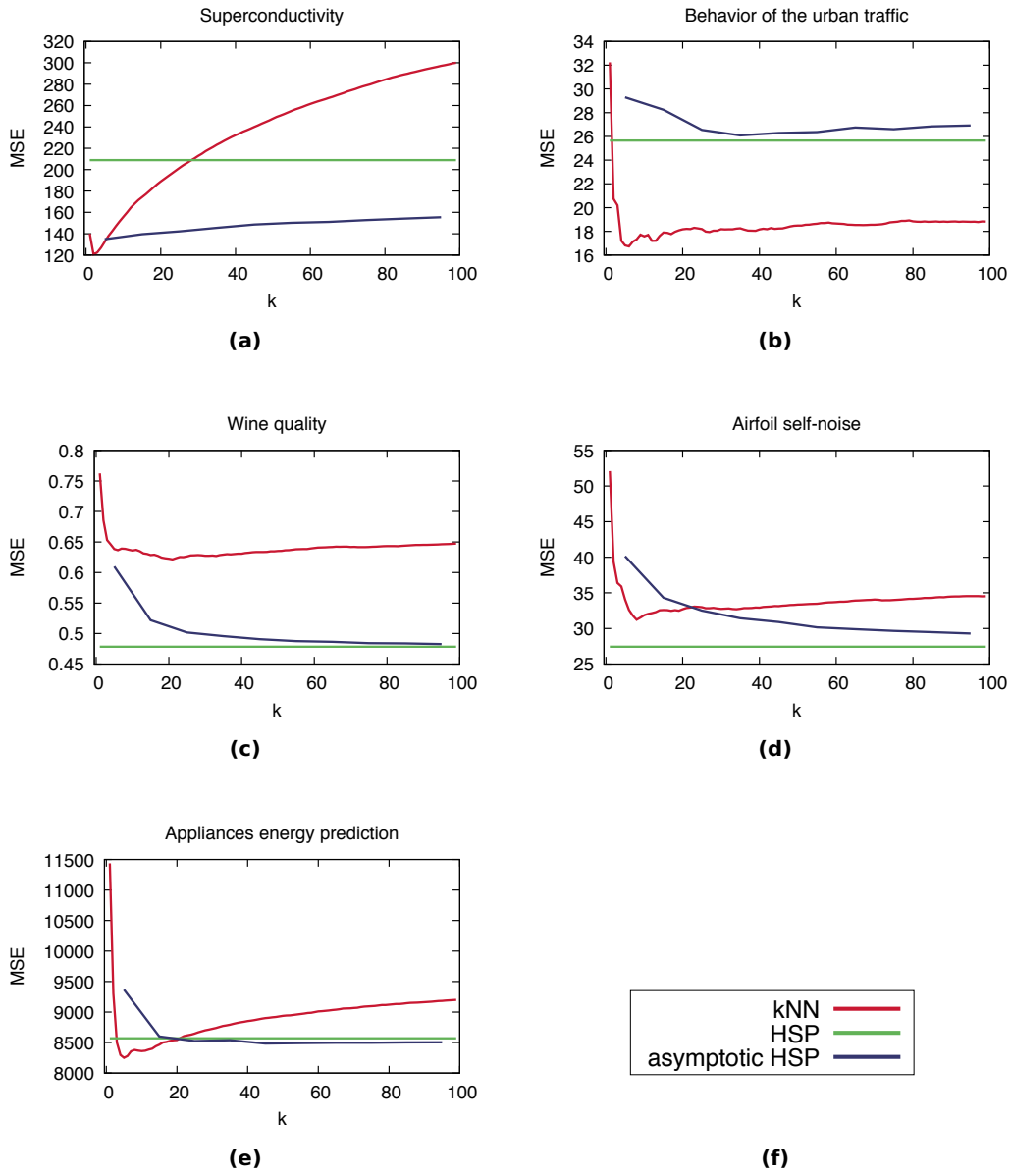


Figura 18. Comparación de error MSE de kNN, HSP y HSP asintótico en problemas de regresión.

Se evaluaron los diferentes algoritmos utilizando el error cuadrático medio (MSE), la raíz del error cuadrático medio (RMSE) y el error absoluto medio (MAE). Las funciones comparan los valores reales con las predicciones obtenidas y las salidas representan la diferencia entre el valor estimado y el valor real según el criterio en cada función. Se utilizaron todos los valores de *k* necesarios para obtener el menor error posible con

kNN y HSP. Además, no fue necesario utilizar las versiones probabilísticas de kNN y HSP debido a que las colecciones de datos son de menor tamaño que las utilizadas en tareas de clasificación.

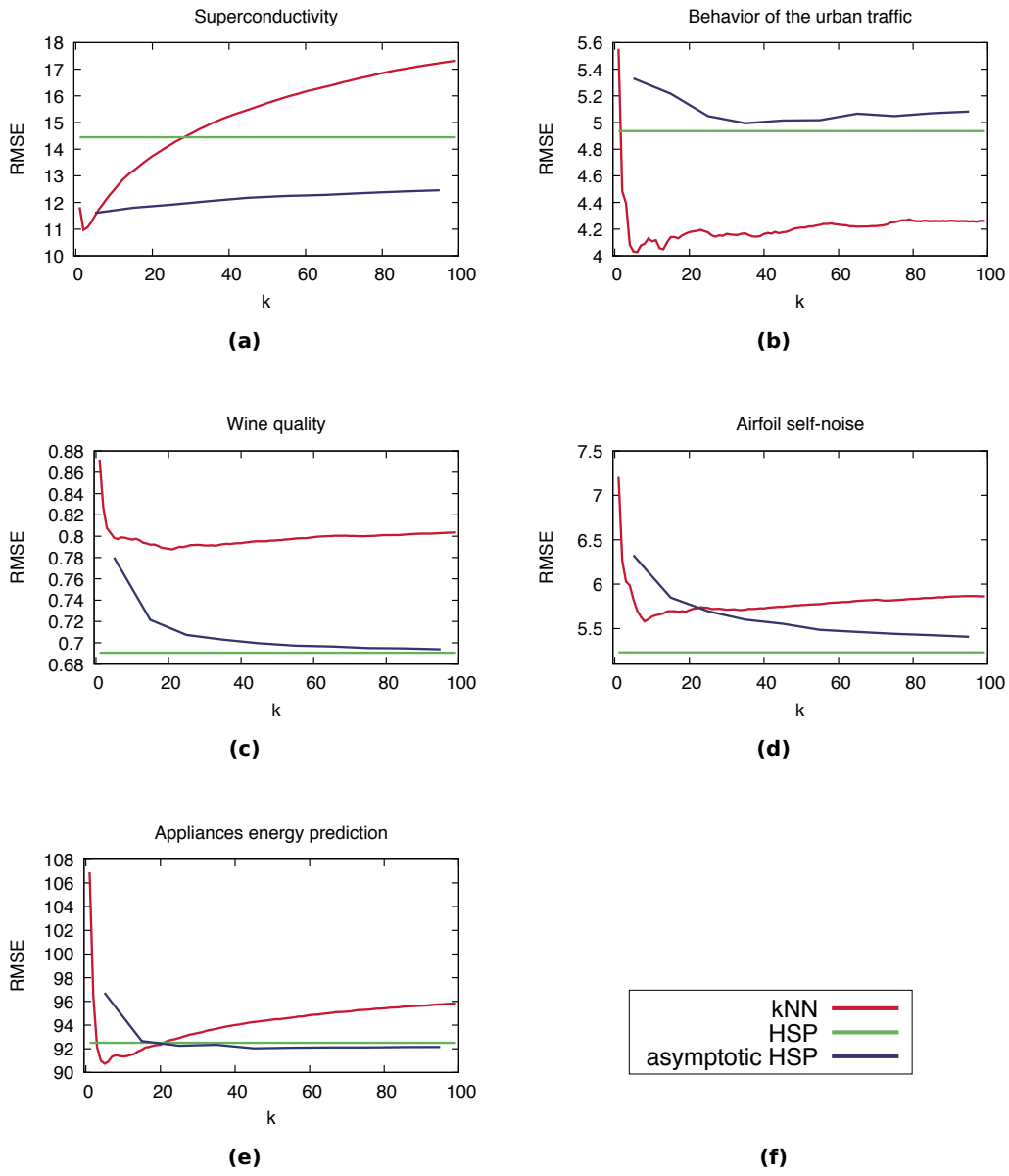


Figura 19. Comparación de error RMSE de kNN, HSP y HSP asintótico en problemas de regresión.

Las gráficas muestran el error correspondiente de la regresión en el eje vertical. Se puede observar que el error de las diferentes técnicas varía en función de *k* con excepción del HSP. El error de la técnica con el algoritmo HSP se mantiene constante porque solo hay un resultado, el cual no depende de hiperparámetros como *k*.

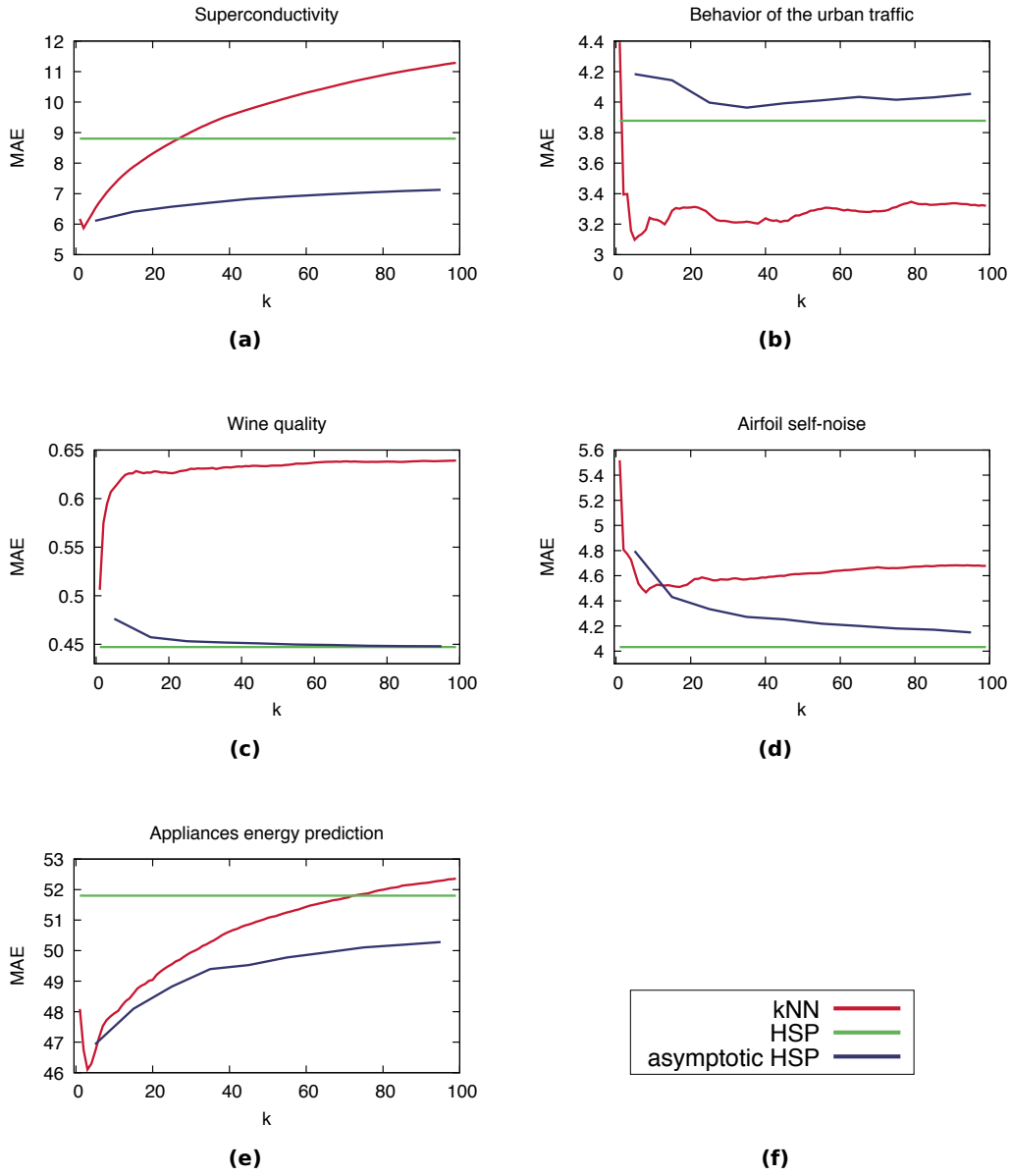


Figura 20. Comparación de error MAE de kNN, HSP y HSP asintótico en problemas de regresión.

Los resultados en MSE y RMSE son similares debido a la relación entre ambas funciones. En estos errores los resultados muestran que no hubo ningún algoritmo que sobresaliera de los demás. El algoritmo kNN tiene mejor desempeño en la colección *Urban traffic*, y para algunos valores de *k* también en las colecciones *Superconductivity* y *Energy*. Por otra parte, el algoritmo HSP tiene mejor desempeño que kNN con todos los valores de *k* en las colecciones *Wine* y *Airfoil*. También, se puede observar que en las colecciones *Superconductivity*, *Traffic* y para algunos valores de *k* en la

colección *Energy*, el HSP asintótico tiene mejor desempeño que el propio HSP.

Cuando se usó MAE, kNN empeoró en la colección *Wine* y HSP empeoró en la colección *Energy*, en relación con los resultados obtenidos con MSE y RMSE. El algoritmo kNN tiene mejor desempeño en la colección *Traffic*, y para algunos valores en las colecciones *Energy* y *Superconductivity*. Por otra parte, el algoritmo HSP tiene mejor desempeño en las colecciones *Airfoil* y *Wine*. En cuanto a la versión asintótica del HSP, se puede observar que tiene mejor desempeño que el propio HSP en *Superconductivity* y *Energy*. En general, al igual que en los errores MSE y RMSE, no hubo ningún algoritmo que sobresaliera de los demás.

Los resultados para los problemas de regresión se resumen en la Tabla 5.

Tabla 5. Error mínimo para cada técnica utilizando los errores: 1. MSE, 2. RMSE, 3. MAE.

		Superconductivity	Traffic	Wine	Airfoil	Energy
1	kNN	**120.67**	**16.72**	0.62	31.23	**8250.86**
	HSP	208.90	25.65	**0.47**	**27.42**	8569.13
	A-HSP	134.82	26.08	0.48	29.28	8484.33
2	kNN	**10.97**	**4.02**	0.78	5.58	**90.74**
	HSP	14.44	4.93	**0.69**	**5.23**	92.5
	A-HSP	11.6	4.99	0.69	5.4	92.03
3	kNN	**5.86**	**3.09**	0.5	4.46	**46.1**
	HSP	8.8	3.87	**0.44**	**4.03**	51.8
	A-HSP	6.1	3.96	0.44	4.14	46.93

Capítulo 6. Discusión

Los algoritmos HSP, HSP asintótico y HSP asintótico probabilístico demostraron tener buen desempeño en tareas de clasificación en colecciones grandes de dimensiones altas. Sin embargo, en tareas de regresión y con colecciones de menor tamaño y menor dimensionalidad obtuvieron un menor desempeño. Las propiedades que brinda el HSP, vecinos diversos entre ellos y a su vez similares a la consulta, se consideran útiles para clasificación, mas no necesariamente para regresión.

Se intentaron (sin éxito) algunas heurísticas para adaptar el HSP a tareas de regresión. Una de estas consistió en fijar un radio definido por la distancia del vecino HSP más lejano. A partir de esto, se obtuvo una nueva vecindad conformada por todas aquellas muestras encontradas dentro de dicho radio. La nueva vecindad se utilizó para llevar a cabo la tarea de regresión. El desempeño de las heurísticas se comparó con el de los algoritmos HSP y kNN con diferentes valores para k; sin embargo, no hubo un algoritmo que sobresaliera del resto; además, la diferencia entre los errores de cada algoritmo no fue significativa.

En el caso del tamaño de las colecciones, el algoritmo kNN ha demostrado mejorar su exactitud cuando se tiene mayor cantidad de datos disponibles, ya que el aprendizaje se basa en comparar las consultas con datos de ejemplo. Dado que el HSP es el mismo tipo de aprendizaje que kNN, obtiene mejor desempeño en colecciones con mayor número de datos. Además, entre mayor sea el tamaño de la colección, se tiene mayor probabilidad de obtener una mejor representación de la consulta, ya que se tienen más muestras para elegir. Como trabajo futuro proponemos obtener un grado de confiabilidad a las predicciones a partir de la densidad local de cada consulta. Un grado alto puede significar mayor centralidad de la muestra y por lo tanto, que está mejor representada, dando mayor confiabilidad en la predicción.

Capítulo 7. Conclusiones y trabajo futuro

En el estado del arte en aprendizaje basado en ejemplos se tiene al algoritmo kNN y otras versiones de este donde se agregan etapas de procesamiento adicional para solucionar sus desventajas. Entre estos últimos se tienen numerosos esfuerzos enfocados en la sintonización del hiperparámetro k, que corresponde al número de vecinos a utilizar para las predicciones. A diferencia del problema de la correcta elección de distancia, la literatura indica que no se obtiene un resultado conclusivo para resolver el problema de elección de k.

En este trabajo se diseñaron tres algoritmos diferentes que superan el estado del arte en tareas de aprendizaje basado en ejemplos: HSP, HSP asintótico y HSP asintótico probabilístico. La característica de estos tres algoritmos es la ausencia de hiperparámetros a sintonizar. Las propiedades del HSP brindan una vecindad diversa y representativa para cada consulta. Además, las versiones asintóticas del HSP incluyen una heurística para reducir la complejidad en tiempo que toma el construir la vecindad del HSP, reduciéndola a la complejidad de kNN indexado.

Las propuestas se compararon con kNN y su versión probabilística. Se utilizó kNN con parámetros óptimos; lo cual es inalcanzable en un entorno real de producción, ya que no hay ninguna justificación teórica para conocer el mejor valor de k. Aún considerando el valor fijo que entrega la mayor exactitud posible en kNN, en todos los casos, los clasificadores kNN a lo más se encontraron a la par del HSP y sus variaciones. Además, se implementaron diferentes reglas de votación para observar el comportamiento de las diferentes vecindades. En todos los casos, la exactitud del HSP y sus variaciones se encontraron sobre la exactitud de kNN y su versión probabilística.

La ausencia de hiperparámetros a sintonizar es importante porque normalmente los clasificadores los utilizan usuarios que no son expertos en clasificación y más bien son expertos en su campo de estudio. El tener un algoritmo con hiperparámetros hace que por lo general se utilicen los valores por defecto, a costa de la eficacia. Cuando se les provee a los usuarios de una herramienta sin hiperparámetros y que además supera al estado del arte, obtienen una herramienta con gran valor para su trabajo.

Un ejemplo similar sucede con los problemas de ordenamiento. Existen distintos algoritmos para obtener el mismo resultado, i.e. los elementos ordenados; sin embargo,

el tiempo de ejecución de cada uno depende del caso específico a resolver. Para hacer una buena elección de algoritmo, la opción más sencilla es utilizar *Quicksort*, que en el caso promedio obtiene el menor tiempo de ejecución. En el caso de kNN y HSP en tareas de clasificación, se tiene la misma complejidad en ambos algoritmos (considerando la versión asintótica del HSP) pero se obtiene una diferencia en la calidad de los resultados. Para un conjunto de datos específico, el desempeño de kNN depende de la elección de k, mientras que en el HSP se tiene la garantía de que el resultado siempre será el mismo, por lo que resulta una buena opción.

Los clasificadores basados en ejemplos sin parámetros podrían ser útiles en muchas aplicaciones; particularmente, en situaciones en las cuales la distribución de los datos varía con el tiempo. Con el enfoque propuesto que no utiliza hiperparámetros como k es posible centrarse en encontrar una función de distancia adecuada para comparar muestras al diseñar un clasificador o una tarea de minería de datos.

Por otro lado, en el caso del problema de regresión, es necesario evaluar nuevas ideas que permitan elegir una vecindad que no necesite hiperparámetros. El problema de regresión es un problema difícil ya que se utilizan muestras finitas para aproximar funciones continuas. Debido a esto, una idea para resolver este problema es aplicar la regresión del algoritmo HSP utilizando un enfoque basado en el método de Monte Carlo. La idea principal del método Monte Carlo es utilizar la aleatoriedad para resolver problemas que simulan fenómenos reales gobernados por una distribución de probabilidad.

El procedimiento general del método Monte Carlo consiste en primeramente definir un dominio de posibles entradas; después, generar entradas de forma aleatoria y repetida a partir de una distribución sobre el dominio; y finalmente, realizar un cálculo determinista sobre las entradas generadas. El procedimiento para utilizar este enfoque para resolver problemas de regresión en combinación con el HSP se ilustra en la Figura 21 y se describe a continuación.

La predicción del valor de un objeto nuevo se puede hacer a partir de objetos generados aleatoriamente alrededor de él. Para esto, se calculan los vecinos HSP de una consulta dada (Figura 21b) y a partir de ellos se define un área para generar objetos aleatorios adentro. Una forma de definir el área es utilizando la envolvente convexa

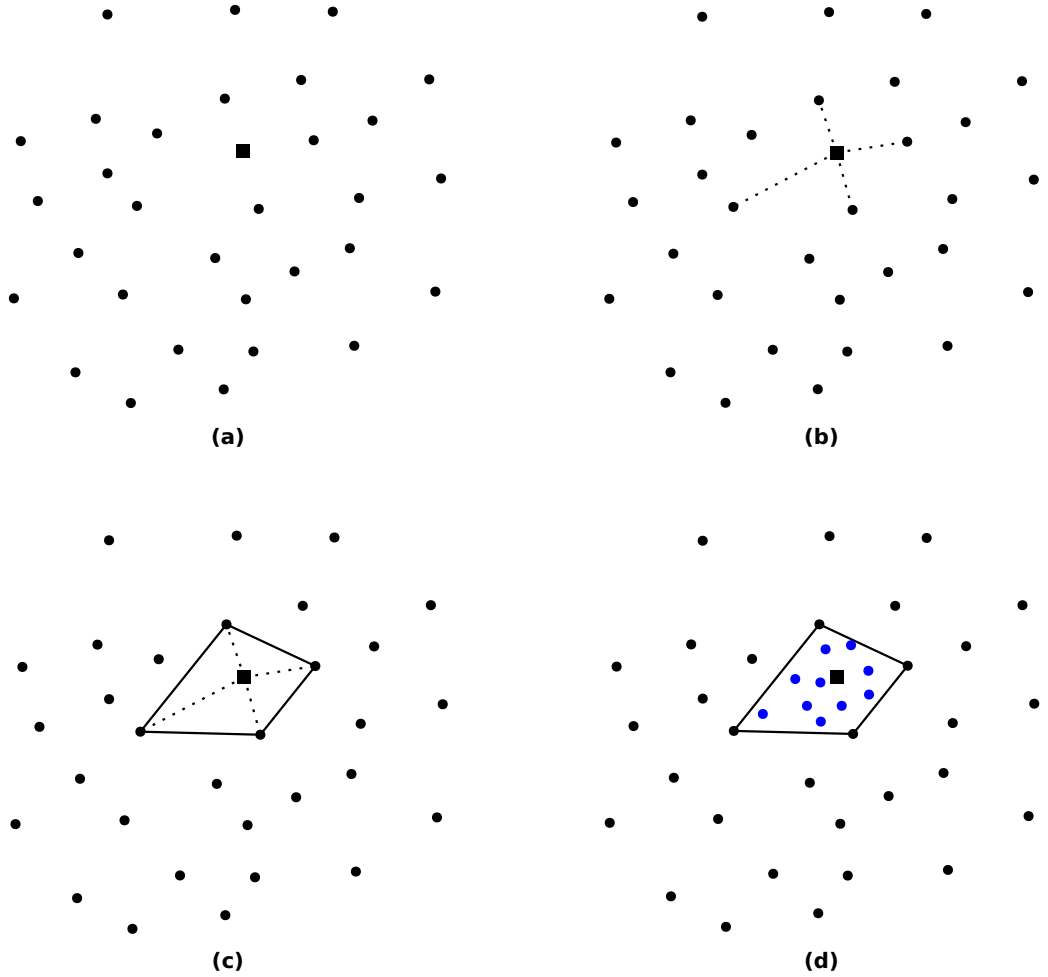


Figura 21. Regresión con HSP y método Monte Carlo

de los vecinos HSP como se muestra en la Figura 21c. Un conjunto S se llama convexo si y solo si, para cualquier par de puntos $p, q \in S$, el segmento pq está contenido completamente en S. Para cualquier subconjunto de S, su envolvente convexa $CH(S)$ es el conjunto convexo más pequeño que pueda contener a S. Una vez obtenida el área, se generan objetos dentro de esta y se utilizan dichos objetos como vecinos de la consulta. Posteriormente, se aplica una regla de votación entre los vecinos y se predice el valor de la consulta.

Calcular la envolvente convexa se vuelve un problema difícil cuando se tienen más de tres dimensiones. Sin embargo, es posible utilizar una aproximación que permita generar puntos aleatorios dentro de la envolvente convexa sin necesidad de calcularla. Para lograr esto, se utilizan los conceptos de combinación convexa y caminata

aleatoria.

Una combinación convexa es una combinación lineal de puntos donde todos los coeficientes son no negativos y suman 1. Todas las posibles combinaciones convexas están dentro de la envolvente convexa de los puntos dados. Dado un conjunto finito de puntos $x_1, x_2, \ldots, x_n$ en un espacio vectorial real, una combinación convexa de esos puntos es un punto que tiene la forma mostrada en la Expresión 23.

$$\alpha_1 x_1, \alpha_2 x_2, \ldots, \alpha_n x_n \tag{23}$$

donde los números reales α_i satisfacen $\alpha_i \geq 0$ y $\alpha_1 + \alpha_2 + \ldots + \alpha_n = 1$. Un punto p generado aleatoriamente dentro de la envolvente convexa estaría definido por las Ecuaciones 24 y 25.

$$p = \alpha_1 v_1 + \alpha_2 v_2 \tag{24}$$

$$\alpha_1 + \alpha_2 = 1 \tag{25}$$

donde v_1 es un vértice semilla dentro de la envolvente convexa (e.g., un vecino HSP) y v_2 un vecino de la consulta, seleccionado aleatoriamente. El valor de $\alpha_1 \in (0, 1)$ también se selecciona aleatoriamente, y α_2 se obtiene despejando la Ecuación 25. El punto p generado se incluye entre los vecinos de la consulta y sustituye al vértice semilla v_1. El procedimiento se repite iterativamente hasta haber generado suficientes puntos. Posteriormente, los vecinos resultantes se utilizan para predecir el valor de la consulta.

Literatura citada

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., y Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Abrahamsen, M., Kleist, L., y Miltzow, T. (2021). Training Neural Networks is ER-complete. *arXiv preprint*. Recuperado el 20 de mayo de 2021 de: `http://arxiv.org/abs/2102.09798`.

Aumuller, M., Bernhardsson, E., y Faithfull, A. (2018). ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms.

Bahety, A. (2014). Extension and Evaluation of ID3-Decision Tree Algorithm. *Entropy (S)*, **2**(1): 1–8.

Biau, G., Chazal, F., Cohen-Steiner, D., Devroye, L., y Rodríguez, C. (2011). A weighted k-nearest neighbor density estimate for geometric inference. *Electronic Journal of Statistics*, **5**.

Biberman, Y. (1994). A context similarity measure. *Lecture Notes in Computer Science*, **784**.

Biswas, N., Chakraborty, S., Mullick, S. S., y Das, S. (2018). A parameter independent fuzzy weighted k-Nearest neighbor classifier. *Pattern Recognition Letters*, **101**: 80–87.

Candanedo, L. M., Feldheim, V., y Deramaix, D. (2017). Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, **140**: 81–97.

Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., Tejeda, H., y Urrutia, J. (2006). Half-Space Proximal: A New Local Test for Extracting a Bounded Dilation Spanner of a Unit Disk Graph. *International Conference On Principles Of Distributed Systems*, pp. 235–245.

Cheng, D., Zhang, S., Deng, Z., Zhu, Y., y Zong, M. (2014). kNN Algorithm with Data-Driven k Value. En: *Advanced Data Mining and Applications*. Springer, Cham, pp. 499–512.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., y Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, **47**(4): 547–553.

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

Cunningham, P. y Delany, S. J. (2020). k-Nearest Neighbour Classifiers 2 nd Edition (with Python examples). *arXiv preprint*. Recuperado el 10 de abril de 2021 de: `https://arxiv.org/abs/2004.04523`.

De Maesschalck, R., Jouan-Rimbaud, D., y Massart, D. (2000). The Mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*, **50**(1): 1–18.

Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li, y Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

Dignam, J. D., Martin, P. L., Shastry, B. S., y Roeder, R. G. (1983). Eukaryotic gene transcription with purified components. *Methods in Enzymology*, **101**(C): 582–598.

Dua, D. y Graff, C. (2017). UCI Machine Learning Repository.

Dudani, S. A. (1976). The Distance-Weighted k-Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, **4**: 325–327.

Ferragina, P. y Navarro, G. (2005). Pizza&Chili Corpus.

Ferreira, R. P. (2016). Combination of Artificial Intelligence Techniques for prediction the Behavior of Urban Vehicular traffic in the city of São Paulo. En: *Anais do 10. Congresso Brasileiro de Inteligência Computacional*, mar. SBIC, Vol. 154, pp. 1–5.

Fix, E. y Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: Small sample performance. Reporte técnico, California Univ Berkeley.

Fukunaga, K. y Hostetler, L. (1973). Optimization of k nearest neighbor density estimates. *IEEE Transactions on Information Theory*, **19**(3): 320–326.

Gautheron, L., Habrard, A., Morvant, E., y Sebban, M. (2020). Metric Learning from Imbalanced Data with Generalization Guarantees. *Pattern Recognition Letters*, **133**.

Ghosh, A. K. (2006). On optimum choice of k in nearest neighbor classification. *Computational Statistics and Data Analysis*, **50**(11).

Google (2021). Colaboratory. Recuperado el 13 de mayo de 2021 de: `https://research.google.com/colaboratory`.

Gramacki, A. (2018). Nonparametric Density Estimation. En: *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Springer, Cham, pp. 7–24.

Hamidieh, K. (2018). A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, **154**: 346–354.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., y Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, **585**(7825): 357–362.

Huh, M., Agrawal, P., y Efros, A. A. (2016). What makes ImageNet good for transfer learning? *arXiv preprint*, pp. 1–10.

Imandoust, S. B. y Bolandraftar, M. (2013). Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events : Theoretical Background. *Int. Journal of Engineering Research and Applications*, **3**(5): 605–610.

Jiang, L., Cai, Z., Wang, D., y Jiang, S. (2007). Survey of Improving K-Nearest-Neighbor for Classification. *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, **1**: 679–683.

Kleinberg, J. M. (1997). Two Algorithms for Nearest-Neighbor Search in High Dimensions. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 599–608.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., y Jupyter Development Team (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. En: F. Loizides y B. Schmidt (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, pp. 87–90.

Kratochvíl, M., Veselý, P., Mejzlík, F., y Lokoč, J. (2020). SOM-Hunter: Video Browsing with Relevance-to-SOM Feedback Loop. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Vol. 11962 LNCS, pp. 790–795.

Krizhevsky, A. y Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images.

Kung, Y. H., Lin, P. S., y Kao, C. H. (2012). An optimal k-nearest neighbor for density estimation. *Statistics and Probability Letters*, **82**(10): 1786–1791.

Lall, U. y Sharma, A. (1996). A nearest neighbor bootstrap for resampling hydrologic time series. *WATER RESOURCES RESEARCH*, **32**(3): 679–693.

Lance, G. N. y Williams, W. T. (1966). Computer Programs for Hierarchical Polythetic Classification ("Similarity Analyses"). *The Computer Journal*, **9**(1): 60–64.

LeCun, Y. y Cortes, C. (2010). MNIST handwritten digit database.

Li, C. y Li, H. (2011). One Dependence Value Difference Metric. *Knowledge-Based Systems*, **24**(5).

Li, L., Weinberg, C. R., Darden, T. A., y Pedersen, L. G. (2002). Gene selection for sample classification based on gene expression data: Study of sensitivity to choice of parameters of the GA/KNN method. *Bioinformatics*, **17**(12): 1131–1142.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., y Lin, X. (2020). Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, **32**(8): 1475–1488.

Loftsgaarden, D. O. y Quesenberry, C. P. (1965). A Nonparametric Estimate of a Multivariate Density Function. *The Annals of Mathematical Statistics*, **36**(3): 1049–1051.

Malkov, Y., Ponomarenko, A., Logvinov, A., y Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, **45**.

Malkov, Y. A. y Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**(4): 824–836.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. pp. 25–25.

Murti, D. M. P., Pujianto, U., Wibawa, A. P., y Akbar, M. I. (2019). K-Nearest Neighbor (K-NN) based Missing Data Imputation. *Proceeding - 2019 5th International Conference on Science in Information Technology: Embracing Industry 4.0: Towards Innovation in Cyber Physical System, ICSITech 2019*, pp. 83–88.

Oquab, M., Bottou, L., Laptev, I., y Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724.

Pan, S. J. y Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**(10): 1345–1359.

Panigrahi, S., Nanda, A., y Swarnkar, T. (2021). A Survey on Transfer Learning. *Smart Innovation, Systems and Technologies*, **194**(10): 781–789.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *Pattern Recognition*, **45**(6): 2041–2049.

Pourhomayoun, M. y Shakibi, M. (2021). Predicting mortality risk in patients with COVID-19 using machine learning to help medical decision-making. *Smart Health*, **20**: 100178.

Qin, Y., Zhang, S., Zhu, X., Zhang, J., y Zhang, C. (2007). Semi-parametric optimization for missing data imputation. *Applied Intelligence*, **27**(1): 79–88.

Shekhar, S. y Xiong, H. (2008). Nearest Neighbor. *Encyclopedia of GIS*, **I**: 771–771.

Simonyan, K. y Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14.

Sitikhu, P., Pahi, K., Thapa, P., y Shakya, S. (2019). A Comparison of Semantic Similarity Methods for Maximum Human Interpretability. *International Conference on Artificial Intelligence for Transforming Business and Society, AITB 2019*.

Song, Y., Liang, J., Lu, J., y Zhao, X. (2017). An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing*, **251**: 26–34.

Sunil, A. y Mount, D. (1993). Approximate nearest neighbor searching. *Proc. 4th Ann. ACMSIAM Symposium on Discrete Algorithms (SODA'93)*, pp. 271–280.

Tang, J., Zhang, X., Yin, W., Zou, Y., y Wang, Y. (2020). Missing data imputation for traffic flow based on combination of fuzzy neural network and rough set theory. *Journal of Intelligent Transportation Systems*, pp. 1–16.

Trebuňa, P., Halčinová, J., Fil'O, M., y Markovič, J. (2014). The importance of normalization and standardization in the process of clustering. *SAMI 2014 - IEEE 12th International Symposium on Applied Machine Intelligence and Informatics, Proceedings*, pp. 381–385.

Vinyals, O., Deepmind, G., Blundell, C., Lillicrap, T., Kavukcuoglu, K., y Wierstra, D. (2016). Matching Networks for One Shot Learning.

Wang, F., Zhen, Z., Wang, B., y Mi, Z. (2017). Comparative Study on KNN and SVM Based Weather Classification Models for Day Ahead Short Term Solar PV Power Forecasting. *Applied Sciences*, **8**(1): 28.

Wang, J., Neskovic, P., y Cooper, L. N. (2007). Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters*, **28**(2).

Wilson, D. R. y Martinez, T. R. (1997). Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, **6**: 1–34.

Wilson, D. R. y Martinez, T. R. (2000). Reduction Techniques for Instance-Based Learning Algorithms. *Machine Learning*, **38**: 257–286.

Wolpert, D. y Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, **1**(1): 67–82.

Wu, X., Kumar, V., Ross, Q. J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z. H., Steinbach, M., Hand, D. J., y Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, **14**(1): 1–37.

Xiang, S., Nie, F., y Zhang, C. (2008). Learning a Mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, **41**(12).

Xiao, H., Rasul, K., y Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.

Yandex, A. B. y Lempitsky, V. (2015). Aggregating local deep features for image retrieval. *Proceedings of the IEEE International Conference on Computer Vision*, **2015 Inter**: 1269–1277.

Zezula, P., Amato, G., Dohnal, V., y Batko, M. (2005). *Similarity Search: The Metric Space Approach*. Springer. p. 170.

Zhang, S., Zong, M., Sun, K., Liu, Y., y Cheng, D. (2014). Efficient kNN Algorithm Based on Graph Sparse Reconstruction. *International Conference on Advanced Data Mining and Applications*.

Zhang, S., Li, X., Zong, M., Zhu, X., y Cheng, D. (2017). Learning k for kNN Classification. *ACM Transactions on Intelligent Systems and Technology*, **8**(3).

Zhang, S., Li, X., Zong, M., Zhu, X., y Wang, R. (2018). Efficient kNN classification with different numbers of nearest neighbors. *IEEE Transactions on Neural Networks and Learning Systems*, **29**(5): 1774–1785.

Zhao, P. y Lai, L. (2020). Analysis of KNN Density Estimation. *arXiv preprint*, pp. 1–35.

Zhu, X., Zhang, S., Jin, Z., Zhang, Z., y Xu, Z. (2011a). Missing value estimation for mixed-attribute data sets. *IEEE Transactions on Knowledge and Data Engineering*, **23**(1): 110–121.

Zhu, X., Zhang, S., Jin, Z., Zhang, Z., y Xu, Z. (2011b). Missing Value Estimation for Mixed-Attribute Data Sets. *IEEE Transactions on Knowledge and Data Engineering*, **23**(1): 110–121.