TESIS DEFENDIDA POR

Martín Gerardo Marmolejo Varela

Y aprobada por el siguiente comité:

Dr. Hugo Homero Hidalgo Silva

Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

Dr. Israel Marck Martínez Pérez

Miembro del Comité

Dr. Jorge Olmos Soto

Miembro del Comité

Dr. José Antonio García Macías

Coordinador del programa en
Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de Estudios
de Posgrado

2 de Julio del 2012.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

COMPARACIÓN DE MÉTODOS KERNEL APLICADOS A LA CLASIFICACIÓN DE OBJETOS REPRESENTADOS COMO GRAFOS

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

Martín Gerardo Marmolejo Varela

Ensenada, Baja California, México.
2012.

RESUMEN de la tesis que presenta **Martín Gerardo Marmolejo Varela**, como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, B. C. Julio del 2012.

COMPARACIÓN DE MÉTODOS KERNEL APLICADOS A LA CLASIFICACIÓN DE OBJETOS REPRESENTADOS COMO GRAFOS

Resumen aprobado por:

Dr. Hugo Homero Hidalgo Silva

Director de Tesis

El reconocimiento de patrones tiene como objetivo el descubrimiento automático de regularidades en datos y mediante el uso de estas regularidades tomar acciones como la clasificación de dichos datos en diferentes categorias. Tiene muchas aplicaciones entre las que se encuentran la minería de datos, detección de fraudes bancarios, reconocimiento de objetos en imágenes, bio-informática, entre otras. Dentro del reconocimiento de patrones, existen métodos llamados *kernels* que se aplican, entre otras cosas, a la clasificación de objetos. Los métodos kernel habían sido aplicados casi exclusivamente en datos con valores reales y pocos tipos de datos especiales, como cadenas o grafos.

Cada vez es más común tener datos no estructurados que necesitan ser representados mediante objetos abstractos como lo son los grafos. Debido a esto, se han propuesto varios métodos kernel para lidiar con el problema de clasificación de grafos para distintos fines, sin embargo, todavía es difícil elegir el método más adecuado para una situación específica.

En este trabajo se estudian y evalúan principalmente dos clases de métodos kernel para grafos del estado del arte conocidos como kernel de caminos más cortos y kernel rápido de subárboles, así como una serie de variantes propuestas. Esto con la finalidad de realizar una comparación entre ellos y determinar el desempeño relativo de los mismos, tanto en porcentaje de clasificación como en tiempo de ejecución.

Se realizaron experimentos con dos tipos de conjuntos de datos: conjuntos de datos reales que incluyen *Mutag*, *PTC* y *Pseudocentros* y conjuntos de datos sintéticos. Resultados experimentales muestran que las variantes que obtuvieron mejores resultados (algunas variantes del kernel de conteo de caminos más cortos y del kernel rápido de sub-árboles sobre grafos de caminos más cortos), a pesar de sus diferencias técnicas, son competitivas entre si ya que ninguna sobresale como la mejor en tiempo de ejecución ni en precisión de clasificación sobre todos los casos de prueba. Además, se puede notar la importancia y utilidad del proceso de conversión, de los grafos originales a grafos de caminos más cortos, ya que este es utilizado por la mayoría de estas variantes.

Palabras clave: Métodos kernel para grafos, aprendizaje de máquina, clasificación.

ABSTRACT of the thesis presented by **Martín Gerardo Marmolejo Varela**, as a partial requirement to obtain the MASTER OF SCIENCE degree in COMPUTER SCIENCES. Ensenada, B. C. July 2012.

COMPARISON OF KERNEL METHODS APPLIED TO THE CLASSIFICATION OF OBJECTS REPRESENTED AS GRAPHS

Abstract approved by:

Dr. Hugo Homero Hidalgo Silva

Thesis Director

The objective of pattern recognition is to automatically discover regularities in data and by using these regularities perform tasks such as the classification of data in different categories. It has many applications in diverse areas such as data mining, fraud detection, object recognition in images, and bio-informatics, to name a few. In pattern recognition, there are approaches known as *kernel methods* that are applied, among other things, to the classification of objects. Kernel methods were initially applied almost exclusively to real valued data and to few special types of data, such as strings or graphs.

It is increasingly common to have unstructured data which need to be represented by abstract objects such as graphs. Because of this, many kernel methods have been proposed to deal with the problem of classification of graphs for different purposes. However, it is still difficult to choose the most appropriate method for a specific situation.

A study and evaluation of state-of-the-art graph kernel methods (shortest path kernel and fast subtree kernel) as well as a series of proposed variations is presented in order to make a comparison between them and determine their relative performance, in both classification rate and computation time.

A series of experiments were performed with two types of datasets: real datasets that include *Mutag*, *PTC* and *Pseudocenters*, and a group of synthetic datasets. Experimental results show that the variants that perform better (some variants of the shortest path count kernel and the fast subtree kernel over shortest path graphs), despite their technical differences, are competitive with each other as none stand out as the best in classification rate or computation time, over all test cases. Furthermore, one can see the importance and usefulness of the conversion process, from the original graphs to shortest path graphs, since it is used by most variants that showed better results.

Keywords: Graph kernel methods, machine learning, classification.

A Lila, Gerardo, Omar, Susana y Susy

Agradecimientos

Con una gran satisfacción y gusto veo hoy terminada una etapa más en mi formación personal y académica. Son muchas las personas que aportaron directa e indirectamente al desarrollo y conclusión de mi trabajo de tesis con quien estoy agradecido. En particula me gustaría agradecer:

A mis padres, Gerardo y Lila, por estar ahí siempre. Gran parte de lo que soy es gracias a su esfuerzo y sacrificio, por eso estoy etérnamente agradecido.

A Omar por ser, además de mi hermano, un amigo incondicional alentándome siempre para seguir adelante.

A mi esposa Susana, por su paciencia, compresión y sobre todo su amor y apoyo incondicional.

A mi hija Susy por ser mi motivación y darme fuerzas en los momentos difíciles.

A mi asesor de tesis, Dr. Hugo Hidalgo, por brindarme su apoyo, consejos y confianza para desarrollar este trabajo.

Al Dr. Carlos Brizuela, por su apoyo, paciencia, consejos y valiosas discusiones que enriquecieron este trabajo.

A los miembros de mi comité de tesis, Dr. Israel Martinez y Dr. Jorge Olmos, por sus valiosas observaciones para contribuir a la mejora de este trabajo.

A mis compañeros de generación y del cubo de *estudiantes de doctorado* con los que pasé muy buenos momentos en el transcurso de mi estancia como estudiante y tesista en el CICESE.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), por brindarme los recursos materiales y académicos para el desarrollo de esta tesis. A los profesores, por haber contribuido con mi formación académica.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por su apoyo económico necesario para realizar mi maestría.

Contenido

Página

Contenido (continuación)

Contenido (continuación)

Lista de Figuras

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Figuras (continuación)

Lista de Tablas

Capítulo I

Introducción

I.1 Antecedentes

El aprendizaje de máquina es una disciplina científica que desarrolla algoritmos o programas para optimizar algún criterio de desempeño usando datos de ejemplo o experiencias pasadas (Alpaydin, 2010). Esto se hace con el entendimiento de que existe algún *proceso* que explica los datos observados de tal manera que, aunque no sea posible identificar este *proceso* en su totalidad, se puede construir una buena aproximación o modelo.

Una vez obtenida esta aproximación o modelo, posiblemente explicando solo una parte de lo observado, se puede utilizar para un mejor entendimiento del propio fenómeno (modelos *descriptivos*) o se pueden usar esos patrones para hacer predicciones (modelos *predictivos*), suponiendo que el comportamiento no cambia demasiado (Alpaydin, 2010).

El aprendizaje de máquina tiene muchas aplicaciones en la actualidad debido, en gran parte, a nuestra habilidad de procesar y recopilar cada vez más información. Las aplicaciones van desde la bio-informática (Sharan y Ideker, 2006), el descubrimiento de medicamentos (Kubinyi, 2003), minería de datos de la red (Washio y Motoda, 2003) y hasta en redes sociales (Kumar *et al.*, 2006).

La *minería de datos* es la aplicación de métodos de aprendizaje de máquina a grandes bases de datos en donde se procesa un gran volumen de información para construir modelos simples para su uso en tareas como la predicción (Alpaydin, 2010). Algunos ejemplos de áreas de aplicación son: detección de fraudes bancarios, la bolsa de valores, diagnóstico médico, optimización de recursos en telecomunicaciones, entre muchos otros.

Otra de las aplicaciones del aprendizaje de máquina es en la inteligencia artificial, en

donde se tiene algún sistema en un ambiente cambiante que la obliga a aprender para adaptarse a las nuevas condiciones. Si el sistema es capaz de afrontar alguna situación y adaptarse, el diseñador del sistema no tiene que preveer y programar soluciones para todas las situaciones posibles.

Una área del aprendizaje de máquina llamada *reconocimiento de patrones* es cuando un sistema debe reconocer el rostro de alguna persona en específico. En este caso, un programa de aprendizaje captura el patrón específico a esta persona y la identifica por medio de este mismo patrón en una imagen dada.

Todos estos datos tienen distintos formatos para su almacenamiento, búsqueda e interpretación. Por ejemplo, en el caso de la predicción de enfermedades, la información puede estar almacenada como una serie de variables binarias representando la existencia o no de síntomas. En este caso, este conjunto de n síntomas se puede representar como un vector de n dimensiones el cual puede ser procesado por los distintos algoritmos de aprendizaje.

Dentro del aprendizaje de máquina, específicamente en el reconocimiento de patrones, existen métodos llamados *métodos kernel* –siendo las *máquinas de vectores soporte* (Cristianini y Shawe-Taylor, 2000) las más populares– que se aplican a la clasificación de objetos entre otras cosas. Estos métodos manipulan los datos, que están representados generalmente como vectores para poder realizar su función.

Sin embargo, cada vez es más común tener datos que son representados como estructuras más flexibles. Un ejemplo de estas estructuras son los grafos, que son estructuras de datos que permiten representar un conjunto de entidades llamadas *vértices* y las relaciones entre ellos por medio de conexiones llamadas *aristas*.

Borgwardt (2007) considera a los grafos como estructuras de datos universales ya que, se puede considerar a cualquier tipo de datos como una instancia de un grafo. Por ejemplo, un escalar puede ser considerado como un grafo con un solo vértice, un vector puede ser considerado como un grafo con un vértice por elemento y una arista entre cada par de elementos consecutivos, etc.

Con esto en mente, Borgwardt (2007), plantea la siguiente pregunta: "¿Por qué los grafos no han sido las estructuras de datos más comunes en las ciencias computacionales por décadas?" La respuesta es que la comparación de grafos es computacionalmente muy costosa.

Mientras que para la comparación entre vectores se tiene como medida de distancia estandar las métricas euclidianas, para los grafos, por ser estructuras más complejas, regularmente de distintos tamaños, no es posible tener una métrica tan simple. Por este motivo, existen muchas formas de intentar desarrollar una función de similitud.

La prueba de *isomorfismo* (descrita en el Capítulo III) ofrece una función de similitud binaria ya que arroja como respuesta si son o no *iguales* dos grafos con respecto a las adyacencias de sus vértices. El problema es que la prueba de isomorfismo es demasiado costosa. Además, la prueba de isomorfismo de subgrafos, que pudiera brindar una medida de similitud más robusta, es NP-difícil (Gärtner *et al.*, 2003).

La aplicación de los métodos kernel había sido mayormente utilizada casi exclusivamente en datos con valores reales y pocos tipos de datos especiales, como cadenas. Kondor y Lafferty (2002) propusieron un método de construcción de familias naturales de kernels en estructuras discretas basado en la idea de la exponenciación de una matriz.

Este trabajo fue uno de los primeros en introducir los kernels en grafos para los que se propuso una clase especial de kernel exponencial llamado kernel de difusión, que está basado en la ecuación de calor.

Desde entonces, se han propuesto muchos métodos kernel para lidiar con el problema de clasificación de grafos. Algunos otros fueron los propuestos por Gärtner *et al.* (2003) y posteriormente por Borgwardt *et al.* (2005). Los llamados kernels marginales de Tsuda *et al.* (2002) fue extendido a grafos por Kashima *et al.* (2003) y posteriormente fue refinado por Mahé *et al.* (2004).

Recientemente, Kondor y Borgwardt (2008), Vishwanathan *et al.* (2010) y Shervashidze *et al.* (2011), por nombrar algunos, ofrecen sus propuestas de kernels para grafos también.

I.2 Definición del problema

Como se ha mencionado, existe un buen número de métodos kernel para grafos en la literatura aplicados para distintos fines. Sin embargo, es difícil saber cuál o cuáles de los métodos son mejores y en qué situaciones específicas. Esto debido principalmente al hecho de que para cada uno de estos métodos, regularmente se utilizan pocos conjuntos de datos en común lo que dificulta la comparación entre ellos.

En este trabajo se pretende evaluar un número de métodos kernels para grafos para así entender el desempeño de los mismos sobrecasos de prueba específicos.

I.3 Objetivo de la investigación

I.3.1 Objetivo general

El objetivo general es determinar el desempeño relativo de un conjunto de métodos kernel aplicados a la clasificación de grafos. El análisis se realizará sobre conjuntos de casos de prueba sintéticos y provenientes de problemas reales.

I.3.2 Objetivos Específicos

1. Definir criterios de desempeño de los métodos kernel para grafos.
2. Seleccionar casos de prueba para la aplicación del experimento.
3. Diseñar un clasificador para la aplicación de métodos kernel para grafos utilizando los casos de prueba seleccionados.
4. Comparar el desempeño de los distintos métodos aplicados en la clasificación de los casos de prueba seleccionados.

I.4 Contribución al conocimiento

En la actualidad, existen trabajos ofreciendo distintas familias de kernels para grafos. La contribución de este trabajo es el hacer un estudio y compilación de las distintas familias de kernels para grafos que existen en la actualidad y reportar un comparativo de sus desempeños tanto en tiempo de ejecución como en la exactitud de la clasificación.

I.5 Metodología

La realización de esta tesis involucró cinco fases, las cuales se describen a continuación:

Fase 1: Entendimiento inicial. Esta fase consiste en el estudio de la literatura sobre los métodos kernel en general y específicamente los métodos kernels para grafos. También involucra el análisis y estudio de los métodos kernel para grafos más utilizados en la literatura para su posible comparación.

Fase 2: Selección de casos de prueba, métodos kernel y criterios de comparación. Esta fase consiste en seleccionar los casos de prueba, determinar que métodos kernel para grafos son los más apropiados para la aplicación a estos casos de prueba y establecer los criterios de comparación entre los métodos kernel seleccionados.

Fase 3: Implementación de kernels para grafos y diseño de un clasificador. Consiste en implementar los métodos kernel para grafos seleccionados en la *Fase 2* así como diseñar el clasificador que se utilizará en los experimentos para poder realizar la comparación entre los métodos kernel en cuestión.

Fase 4: Experimentos y análisis de resultados. Esta fase consiste en realizar los experimentos con los casos de prueba seleccionados en la *Fase 2* con cada uno de los métodos kernel implementados. Posteriormente se analizan los resultados de cada uno de estos métodos para su comparación.

Fase 5: Redacción de la tesis. Se concluye la investigación mediante la redacción de la tesis, de acuerdo a las actividades que se realizaron en las etapas anteriores, además de incluir las conclusiones, aportaciones y trabajo futuro que se deriva de este trabajo de investigación.

I.6 Organización de la tesis

Este trabajo consta de seis capítulos los cuales están organizados de la siguiente manera:

El Capítulo II "Kernels" introduce el problema de representación de datos con estructuras complejas y explica el concepto de *kernel* aplicado a cualquier tipo de objeto. Además aborda el tema de construcción de métodos kernel válidos y por último se habla de uno de los métodos kernel más populares en la actualidad, las *SVM*'s o *máquinas de vectores soporte*.

En el Capítulo III "Kernels para grafos" se trata de cómo se pueden aplicar los métodos kernel cuando los objetos a clasificar son grafos. Para esto primero se introducen conceptos básicos de la teoría de grafos y posteriormente se introducen tres de los métodos más utilizados: *caminatas aleatorias*, *caminos más cortos* y *subárboles*. Por último se introducen una serie de variaciones a estos métodos buscando mejorar el desempeño ya sea en tiempo de ejecución o en exactitud de clasificación.

En el Capítulo IV "Experimentos y resultados" se describen brevemente los detalles de los experimentos realizados en este trabajo así como los conjuntos de datos que fueron utilizados para conducirlos. Además, se exponen los resultados de los experimentos con conjuntos de datos reales (*Mutag*, *PTC* y *Pseudocentros*) y se muestra una parte de los resultados de los experimentos con datos sintéticos.

En el Capítulo V "Mejores resultados", se realiza un análisis separado por tipo de conjunto de datos utilizados y se seleccionan los métodos con mejor desempeño por cada una de las dos familias en cada experimento y se comparan de forma más general. Además,

se incluyen algunos de los mejores resultados reportados en la literatura en la clasificación de datos reales para evaluar el desempeño relativo con respecto a estos.

En el Capítulo VI "Conclusiones y trabajo futuro" se discuten los resultados obtenidos y se comentan lo retos y oportunidades que se presentan como trabajo futuro.

En el Apéndice A se muestran los resultados del resto de los experimentos realizados con datos sintéticos descritos en el Capítulo IV.

Por último, en el Apéndice B se describen a detalle todos los conjuntos de datos utilizados en el trabajo. Esto incluye los conjuntos de datos reales (*Mutag*, *PTC* y *Pseudocentros*) así como los conjuntos de datos sintéticos.

Capítulo II

Kernels

II.1 El problema de la representación de datos

Primeramente se define a un conjunto de n objetos a analizar como $S = x_1, x_2, ..., x_n$. Suponemos que cada objeto x_i es un elemento del conjunto de todos los objetos a analizar $\mathcal{X}$ (imágenes, moléculas, grafos, etc). Para poder diseñar métodos que analicen estos objetos, primeramente se debe de poder representar el conjunto S para un procesamiento posterior, la pregunta es cómo hacerlo.

Normalmente, los métodos no basados en kernels primero definen una representación para cada objeto y después representan al conjunto de objetos como el conjunto de representaciones. Formalmente, esto significa que una representación $\phi(x) \in F$ es definida para cada objeto $x \in \mathcal{X}$, en donde cada representación puede ser un vector de valores reales ($F = \mathbb{R}^p$), una cadena de longitud finita (F será entonces el conjunto de todas las cadenas de longitud finita), o alguna otra representación más compleja. El conjunto S es representado como el conjunto de representaciones de objetos individuales $\phi(S) = (\phi(x_1), \phi(x_2), ..., \phi(x_n))$ (Vert *et al.*, 2004).

Por otro lado, los métodos kernels no representan a los objetos individualmente sino a través de un conjunto de comparaciones por pares (ver Figura 1). En otras palabras, en vez de usar un mapeo $\Phi : \mathcal{X} \to F$ para representar cada objeto $x \in \mathcal{X}$ por $\phi(x) \in F$, se usa una comparación de valores reales $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ y el conjunto S es representado por la matriz de $n \times n$ de comparaciones por pares $k_{i,j} = k(x_i, x_j)$ (Vert *et al.*, 2004).

Se pueden hacer tres observaciones importantes acerca de las propiedades de los kernels:

1. La representación en forma de matriz de $n \times n$ es independiente de la naturaleza

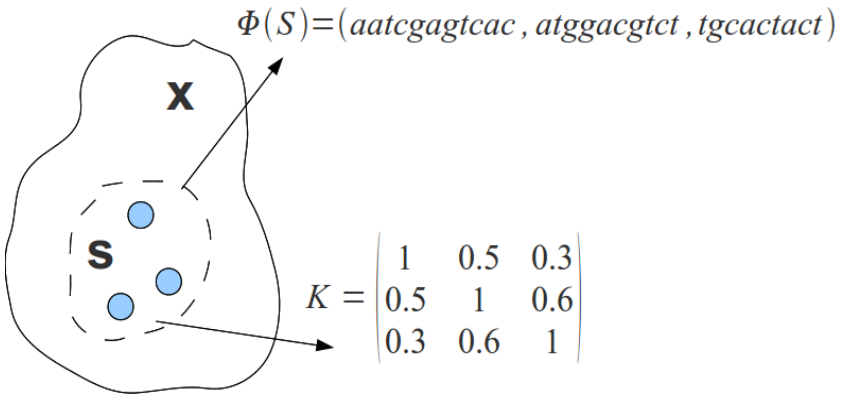


Figura 1. Representación de datos habitual vs kernels.

del objeto a ser analizado. De tal manera que pueden ser imágenes, moléculas o secuencias y la representación siempre será una matriz cuadrada con valores reales.

2. La matriz usada para representar los n objetos siempre será de tamaño $n \times n$ sin importar la naturaleza o complejidad de dichos objetos.

3. Existen muchos casos en los que la comparación de objetos es mucho más sencilla que encontrar una representación explícita para cada uno de los objetos que un algoritmo dado puede procesar (Vert *et al.*, 2004).

Dentro de los beneficios más importantes de estos puntos tenemos que, mientras se haya definido una función k válida, un mismo algoritmo para procesar una matriz con dichas características puede utilizarse para analizar distintos tipos de objetos. Además, permite una modularidad entre el diseño de un algoritmo para representar los objetos y un algoritmo para procesar los datos representados.

II.2 Definición general

La función de comparación k es un componente crítico de cualquier método kernel ya que define cómo el algoritmo "ve" los datos. Muchos de los métodos kernel pueden procesar

solo matrices cuadradas, simétricas y definidas positivas. Esto significa que si k es una matriz de $n \times n$ comparaciones por pares, debería de satisfacer $k_{i,j} = k_{j,i}$ para cualquier $1 \leq i, j \leq n$, y $c^T kc \geq 0$ para cualquier $c \in \mathbb{R}$.

Esto motiva la siguiente definición (Vert *et al.*, 2004):

Definición 1. Una función $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ es llamada *kernel definido positivo* si y solo si es:

1. **simétrica**, esto es, si $k(x, x') = k(x', x)$ para cualquier par de objetos x y $x' \in \mathcal{X}$,

2. **definida positiva**, esto es,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0,$$

para cualquier $n > 0$, cualquer selección de n objetos $x_1, x_2, \cdots, x_n \in \mathcal{X}$ y cualquier selección de números reales $c_1, c_2, \cdots, c_n \in \mathbb{R}^n$.

Debido a que en este trabajo solo se manejan kernels *definidos positivos*, por simplicidad, de este punto en adelante se les hará referencia solo como *kernels*.

Imponer la condición de que la función de comparación sea un kernel viene con un costo, ya que restringe la clase de funciones que se pueden utilizar. Sin embargo, permite el uso de los métodos kernel lo que compensa el costo de dicha limitante.

II.3 Kernels como producto interno

Para lograr un mejor entendimiento del concepto es necesario establecer un ejemplo sencillo el cual lleve a una propiedad fundamental de los kernels. Suponemos que los datos a analizar son vectores reales, esto es, $\mathcal{X} = \mathbb{R}^p$ y cualquier objeto es escrito como $x = (x_1, x_2, \cdots, x_p)^T$. Si comparamos tales vectores usando el producto interno, para cada $x, x' \in \mathbb{R}^p$, tenemos:

$$k_{lineal}(x, x') = x^T x' = \sum_{i=1}^{p} x_i x'_i. \tag{1}$$

Esta función es un kernel ya que es simétrico ($x^T x' = x'^T x$), y es definido positivo como resultado de este cálculo, válido para cualquier $n > 0$, $x_1, \cdots, x_n \in \mathbb{R}^p$, y $c_1, \cdots, c_n \in \mathbb{R}$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_L(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j x_i^T x_j = \| \sum_{i=1}^{n} c_i x_i \|^2 \geq 0$$

Este kernel es llamado *kernel lineal* y es simplemente el producto interno entro dos vectores. La limitante obvia de este kernel es el hecho de que es necesario tener los datos en forma de vectores. Para objetos más generales $y \in \mathcal{Y}$ con $\mathcal{Y}$ un conjunto de objetos dado, se puede adaptar una manera sistemática de definir kernels, esto es, primero representando a cada objeto en cuestión $y \in \mathcal{Y}$ como un vector $\phi(y) \in \mathbb{R}^p$, y después definiendo un kernel para cualquier $y, y' \in \mathcal{Y}$ de esta manera:

$$k(y, y') = \phi(y)^T \phi(y') \tag{2}$$

Cualquier mapeo $\phi :\rightarrow \mathbb{R}^p$ para algún $p \geq 0$ resulta en un kernel válido a través de (1).

Teorema 1. *Para cualquier* kernel k *en un espacio* $\mathcal{X}$*, existe un espacio de Hilbert* $\mathcal{F}$ *y un mapeo* $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ *tal que*

$$k(x, x') = \langle \phi(x), \phi(x') \rangle, \quad \textit{para cualquier } x, x' \in \mathcal{X} \tag{3}$$

en donde $\langle u, v \rangle$ *representa el producto interno en el espacio de Hilbert entre dos puntos* $u, v \in \mathcal{F}$*.*

Esto nos muestra que los kernels pueden ser pensados como productos internos en algún espacio $\mathcal{F}$, llamado espacio de características (Figura 2). Es por esto que el uso de los kernels se reduce a representar cada objeto $x \in \mathcal{X}$ como un vector $\phi(x) \in \mathcal{F}$, y calcular productos punto.

En este caso, a diferencia de la representación explícita de los objetos como vectores, no es necesario calcular cada uno de los puntos en el conjunto de datos S ya que solo son necesarios los productos punto por pares. De hecho, hay muchos casos en los que el espacio

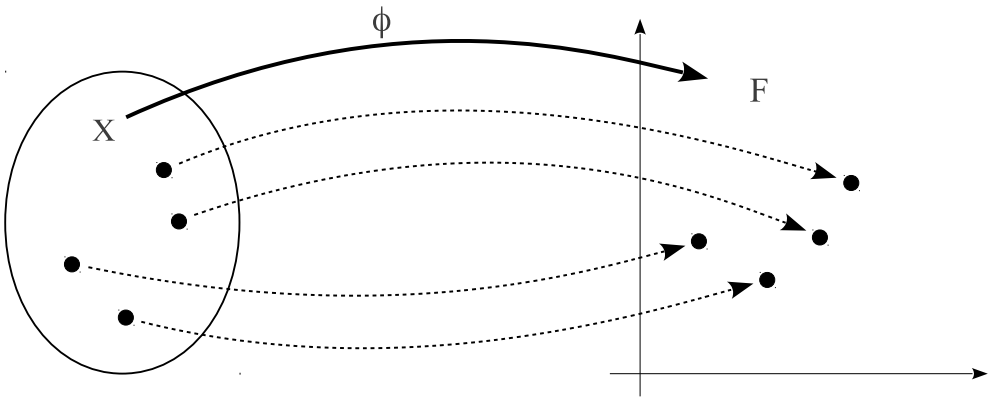


Figura 2. Cualquier kernel en un espacio X puede ser representado como un producto interno después del mapeo del espacio X al espacio de Hilbert F llamado espacio de características.

de características asociado con un kernel simple es de dimensiones infinitas y por ende la imagen $\phi(x)$ de un punto x es complicado de representar aunque el kernel sea simple de calcular (Vert *et al.*, 2004).

Con esta idea de los kernels como producto punto se les puede dar una interpretación geométrica a los puntos $\phi(x) \in \mathcal{F}$ siendo la representación de los puntos $x \in \mathcal{X}$ en el espacio de características. Se le puede dar un sentido de medida de similitud a los kernels, de tal manera que un valor grande de $k(x, x')$ quiera decir que los puntos x y $x' \in \mathcal{X}$ son *parecidos* entre si y vice versa.

II.3.1 El truco del kernel

El truco del kernel es un principio general y simple basado en la propiedad de los kernels de que pueden ser pensados como productos internos. Es expresado por Vert *et al.* (2004) de la siguiente manera:

> "Cuaquier algoritmo para datos vectoriales que puede ser expresado solamente en términos de productos punto entre vectores puede ser implícitamente realizado en el espacio de características asociado con cualquier kernel, al reemplazar

cada producto punto por una evaluación del kernel."

El truco del kernel puede parecer sencillo pero tiene muchas consecuencias prácticas. Primeramente es una manera muy conveniente de transformar métodos lineales, como el análisis lineal discriminante (Hastie *et al.*, 2001) o el análisis de componentes principales (Jolliffe, 1986), a métodos no lineales simplemente reemplazando el producto punto clásico por un kernel. De esta manera, se consigue la *no linealidad* sin ningún costo adicional, ya que el algoritmo se mantiene igual. La operación que transforma un algoritmo lineal a un método kernel más general es conocida como *kernelización*.

Segundo, la combinación del truco del kernel con kernels definidos sobre datos no vectoriales (documentos, grafos, etc) permite la aplicación de muchos algoritmos clásicos (sobre vectores) a virtualmente cualquier tipo de dato, siempre y cuando se pueda definir un kernel.

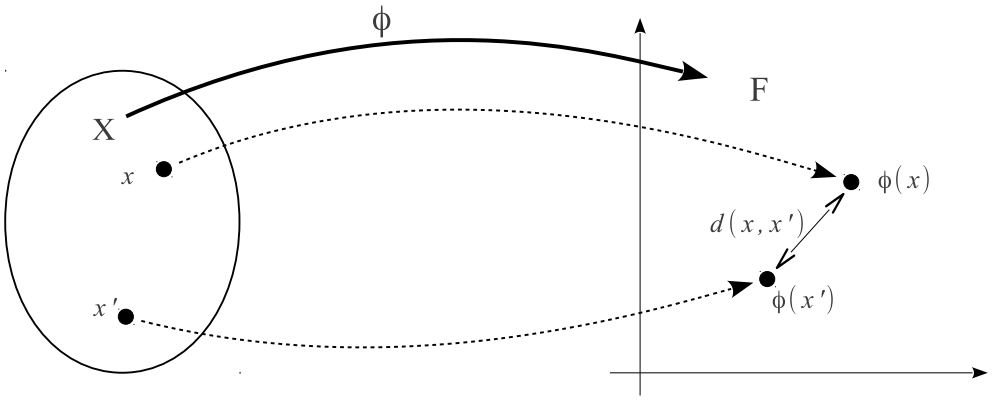


Figura 3. Dado un espacio X dotado con un kernel, se puede definir una distancia entre dos puntos en X mapeados a un espacio de características F asociado con dicho kernel. Esta distancia puede ser calculada sin saber explícitamente el mapeo ϕ gracias al truco del kernel.

II.3.2 Construcción de Kernels

Para poder aprovechar el truco del kernel es necesario poder construir funciones kernel válidas. Una de las formas de hacerlo es seleccionando el mapeo al espacio de características

$\phi(x)$ para encontrar el kernel correspondiente.

Otra alternativa es construir la función kernel directamente. En este caso, se debe asegurar que la función que se escoja sea un kernel válido, es decir, que corresponda a un producto escalar en algún espacio de características.

Es por esto que se necesita una manera de probar si una función kernel es válida o no sin necesidad de construir la función explícitamente. De acuerdo a Shawe-Taylor y Cristianini (2004), una condición necesaria y suficiente para que una función $k(x, x')$ sea un kernel válido es que la matriz de cuyos elementos son dados por $k(x_n, x_m)$ (matriz de Gram $\mathbf{K}$), deben ser positivos definidos para cualquier conjunto de elecciones del conjunto $\{x_n\}$.

Una técnica muy poderosa de construir kernels nuevos es diseñar el kernel como una combinación de otros kernels. Es decir, diseñar un kernel más complejo mediante el uso de kernels que por si solos sean más simples. Todo depende de qué es lo que se desea y las características del problema a resolver.

Existen muchos tipos de kernels básicos que se utilizan para este tipo de propósitos. Algunos de los más populares se muestran a continuación. Para un los objetos x y x', que en este caso son vectores:

$$k_{Dirac}(x, x') = \begin{cases} 1 & si\ x = x', \\ 0 & otro\ caso, \end{cases} \tag{4}$$

$$k_{RBF}(x, x') = \exp\left(\frac{-\|x - x'\|^2}{2\sigma^2}\right), \tag{5}$$

$$k_{Browniano}(x, x') = \max(0, c - |x - x'|), \tag{6}$$

$$k_{poli}(x, x') = (x^T x + 1)^p. \tag{7}$$

El kernel *Dirac* o *delta* (4) es una función binaria la cual evalua una sola condición (en este caso es la igualdad de sus parámetros) y regresa como resultado 1 en caso de cumplirse

y 0 en caso contrario. Por otro lado, el kernel *Gaussiano RBF* (5), al igual que el anterior, regresa un 1 en caso de que los parámetros sean iguales, pero en caso de no ser así regresa un valor menor a 1 pero distinto de 0 en donde σ es un parámetro que controla la resolución de este rango. El kernel de *Puente Browniano* (6), regresa como resultado el máximo valor cuando los dos parámetros son iguales y 0 a aquellos valores que sobrepasan una diferencia mayor que una constante c.

Estos cuatro kernels válidos son ampliamente utilizados individualmene o como parte de kernels más complejos.

II.3.2.1 Técnicas para la construcción de kernels

Construir kernels usando otros kernels más simples como bloques de contrucción es una técnica poderosa y flexible. Para este propósito, se pueden usar una serie de propiedades contenidas en la Tabla I.

Tabla I. Técnicas para la construcción de kernels.

Dados los kernels válidos $k_1(x, x')$ y $k_2(x, x')$, los siguientes kernels también son válidos:

$$\begin{aligned}
k(x,x') &= ck_1(x,x') \\
k(x,x') &= f(x)k_1(x,x')f(x') \\
k(x,x') &= q(k_1(x,x')) \\
k(x,x') &= \exp(k_1(x,x')) \\
k(x,x') &= k_1(x,x') + k_2(x,x') \\
k(x,x') &= k_1(x,x')k_2(x,x') \\
k(x,x') &= k_3(\phi(x),\phi(x')) \\
k(x,x') &= x^T A x' \\
k(x,x') &= k_a(x_a,x'_a) + k_b(x_b,x'_b) \\
k(x,x') &= k_a(x_a,x'_a)k_b(x_b,x'_b)
\end{aligned}$$

donde $c > 0$ es una constante, $f(\cdot)$ es cualquier función, $g(\cdot)$ es un polinomio con coeficientes no negativos, $\phi(x)$ es una función que va de x a $\mathbb{R}^M$, $k_3(\cdot,\cdot)$ es un kernel válido en $\mathbb{R}^M$. Una matriz simétrica positiva semidefinida A, x_a y x_b son variables (no necesariamente distintas) con $x = (x_a, x_b)$, y k_a y k_b son funciones de kernel válidas en sus espacios respectivos (Tabla extraida de Bishop (2006), 296).

II.4 Métodos Kernel

Se puede definir a un método kernel como aquel en el cual los datos a analizar solo entran al algoritmo a través de la función kernel. En otras palabras, los métodos kernels son algoritmos que toman como entrada la matriz de similitud definida por un kernel.

Históricamente, el primer método kernel reconocido como tal lo conforman las *Máquina de vectores soporte* (Boser *et al.*, 1992) o *SVM*(Support Vector Machines, en inglés).

En esta sección se describirá este método pues es una parte importante de este trabajo al utilizarlo para la clasificación en los experimentos realizados. Sin embargo, no se cubrirá exhaustivamente el tema ya que existen muchos recursos disponibles para su estudio como lo son (Noble, 2006) y (Cristianini y Shawe-Taylor, 2000).

II.5 Máquinas de vectores soporte

Las máquinas de vectores soporte constituyen un método de aprendizaje supervisado[1], en el contexto del reconocimiento de patrones y el aprendizaje automático (Lewis *et al.*, 2006).

El concepto del *kernel* fue introducido al campo del reconocimiento de patrones por Aizerman *et al.* (1964) en el contexto del método de funciones de potencia, llamado así debido a una analogía con la electrostática.

Después de muchos años fue reintroducido al aprendizaje de máquina por Boser *et al.* (1992) en el contexto de los clasificadores de márgenes máximos dando auge a la técnica de *máquinas de vectores soporte*.

Una propiedad importante de las máquinas de vectores soporte es que la determinación de los parámetros modelo corresponde a un problema de optimización convexa, y por lo tanto cualquier solución local es también un óptimo global (Bishop, 2006).

[1] **aprendizaje supervisado**: algoritmo en donde se busca aprender un mapa de una entrada a una salida en donde los valores correctos son proporcionados por un *supervisor* (Alpaydin, 2010).

II.5.0.2 Hiperplano óptimo de decisión

Las máquinas de vectores soporte fueron ideadas en un principio para la resolución de clasificación binaria con clases linealmente separables. También se les conocía como "hiperplano óptimo de decisión" debido a que la solución es aquella que clasifica correctamente las muestras colocando un hiperplano de separación lo más lejos posible de todas ellas. A las muestras más cercanas al hiperplano óptimo se les conoce como *vectores soporte* (ver Figura 4).

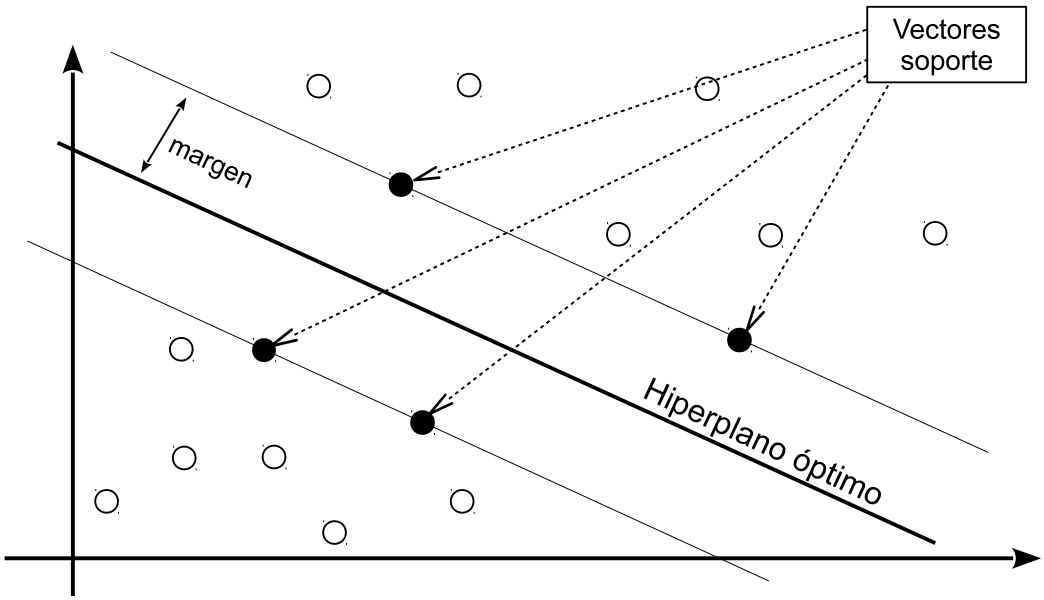


Figura 4. Hiperplano óptimo y vectores soporte. Entrenar una máquina de vectores soporte consiste en encontrar el hiperplano óptimo, es decir, el que maximice la distancia entre los patrones más ceranos.

El hiperplano óptimo de decisión es buscado en el espacio de decisión (espacio de características) al que fueron transformados previamente los datos (ver Figura 5). Para realizar esta transformación sólo es necesario conocer el kernel, con lo que simplifica la obtención de funciones de decisión no lineales.

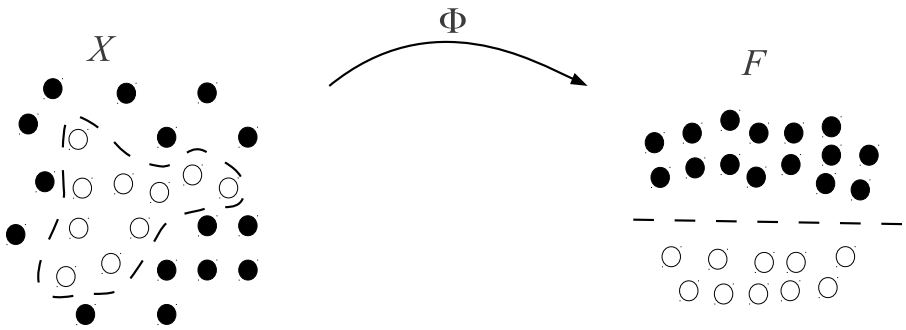


Figura 5. Separación en el campo de características.

II.5.0.3 Funcionamiento de las máquinas de vectores soporte

El funcionamiento de las SVM, a grandes razgos, consiste en la transición del problema original a un espacio de mayor dimensionalidad mediante funciones kernel, para tener mayores probabilidades de que las clases sean linealmente separables en el espacio de características.

Mucho del poder de estas máquinas viene de su criterio para seleccionar el plano de separación cuando hay varios planos candidatos: la máquina de vectores soporte elegirá el plano que mantenga un margen máximo entre los puntos de entrenamiento (Zaki *et al.*, 2009). El problema de dicha elección puede ser formulada y resuelta mediante programación cuadrática (Schölkopf y Smola, 2002).

Un beneficio importante de las máquinas de vectores soporte sobre otros métodos es la complejidad del clasificador, obtenido mediante este enfoque, la cual se caracteriza por un número de vectores soporte independientemente de la dimensionalidad del espacio de tranformación (Duda *et al.*, 2001).

La aplicación de las máquinas de vectores soporte a un problema de clasificación consta de dos fases: entrenamiento y predicción (o prueba). En el entrenamiento, el conjunto de datos de entrada debe tener asociada una etiqueta binaria, normalmente +1 para la clase positiva y -1 para la clase negativa. El algoritmo de entrenamiento buscará un hiperplano

que separe los ejemplos positivos de los negativos.

En la etapa de predicción, la máquina de vectores soporte tratará de establecer cuáles etiquetas corresponden a los datos del nuevo conjunto de entrada, es decir, determinar en qué lado del hiperplano de separación caen los datos (Lewis *et al.*, 2006).

Supóngase que el conjunto de datos $\mathcal{S}$ consiste de una serie de objetos $x_1, x_2, \cdots, x_n \in \mathcal{X}$, junto con una serie de etiquetas $y_1, y_2, \cdots, y_n \in \mathcal{Y}$, donde $y_i \in \{+1, -1\}$ (en este caso), asociadas con los objetos. Las SVMs son métodos kernel para aprender una función $f : \mathcal{X} \to \mathcal{Y}$ de $\mathcal{S}$, que puede ser utilizada para predecir la etiqueta de cualquier objeto nuevo $x \in \mathcal{X}$ por $f(x)$ (Vert *et al.*, 2004).

Primeramente se considera cuando los objetos que recibe el algoritmo son vectores, $\mathcal{X} \subseteq \mathbb{R}^p$. En este caso, la *SVM* trata de separar las dos clases de puntos usando una función lineal de la forma

$$f(x) = \text{signo}(\langle w, x \rangle + b), \tag{8}$$

con $w \in \mathbb{R}^p$ y $b \in \mathbb{R}$ (Vert *et al.*, 2004).

Dicha función asigna una etiqueta de +1 a los puntos $x \in \mathcal{X}$ con $f(x) \geq 0$ y la etiqueta de -1 a los puntos $x \in \mathcal{X}$ con $f(x) < 0$. El hiperplano de separación está dado por (Leslie *et al.*, 2002):

$$\{x \in \mathbb{R}^n : \langle w, x \rangle + b = 0\}, \tag{9}$$

donde w es un vector normal al hiperplano y b es el parámetro de desplazamiento.

Para una función candidata $f(x) = \langle w, x \rangle + b$ uno puede verificar si para cada observación (x_i, y_i), f la clasificó correctamente, es decir, si $y_i f(x_i) \geq 0$ o no (Vert *et al.*, 2004).

Un punto clave de las máquinas de vectores soporte es que la optimización se simplifica resolviendo el problema dual de optimización minimizando $\|w\|^2/2$, sujeto a $y(w^T x + b) \geq 1$, la solución a (9) se encuentra resolviendo este problema.

Para el caso linealmente separable, el clasificador de margen máximo es encontrado mediante la solución de los *pesos* óptimos α_i, $i = 1, \cdots, m$ en el problema dual (Leslie

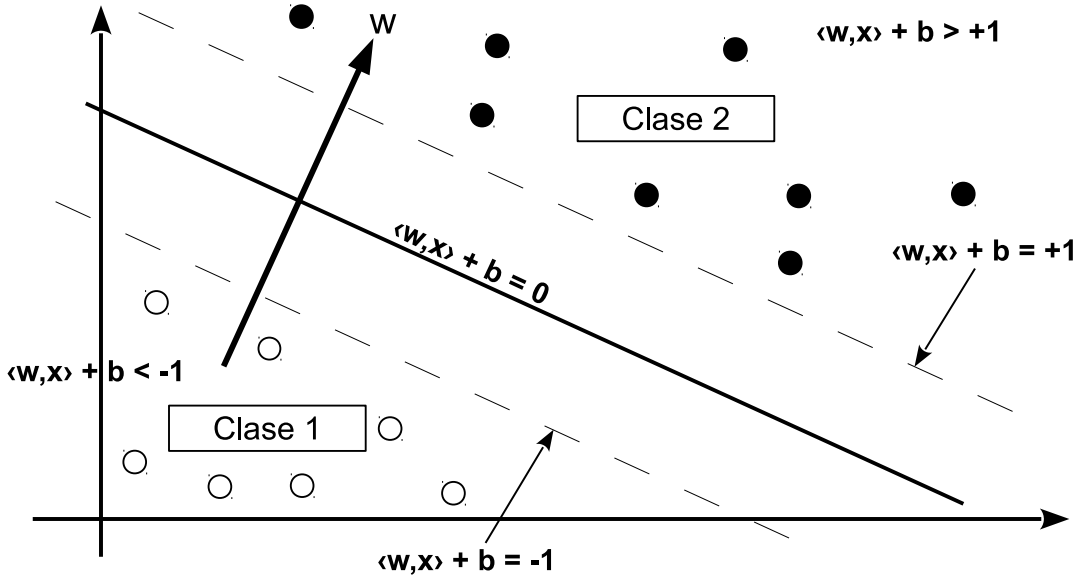


Figura 6. Separación de clases. Se puede observar la separación de clases mediante el hiperplano óptimo, procurando maximizar la distancia entre los vectores soporte.

et al., 2002):

$$\text{Maximizar} \left\{ \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \right\}, \tag{10}$$

sujeta a $\alpha_i \geq 0$, $\sum_i \alpha_i y_i = 0 \, \forall \, i$.

Entonces los parámetros (w, b) del clasificador son determinados por los α_i óptimos, los datos de entrenamiento y la función resuelta $f(x) = \sum_i \alpha_i \langle x_i, x \rangle + b$.

De (9) podemos ver que el aprender un clasificador lineal y predecir la clase de un nuevo punto solo involucra los puntos del conjunto de entrenamiento a través de sus productos punto. Es por esto que se puede aplicar el truco del kernel para realizar el algoritmo de SVM en un espacio de características asociado con un kernel general. El algoritmo se puede declarar de la siguiente manera (Vert *et al.*, 2004):

Encontrar $\alpha = (\alpha_1, \cdots, \alpha_n)$ que maximiza

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j k(x_i, x_j), \tag{11}$$

sujeta a $\alpha_i \geq 0$, $\sum_i \alpha_i y_i = 0 \, \forall \, i$

Según Noble (2006), la esencia de las máquinas de vectores soporte puede ser entendida sin el uso de fórmulas, para lo cual propone el conocimiento de cuatro conceptos básicos: el *hiperplano de separación*, el *hiperplano óptimo*, el *margen suave* y la *función kernel*.

El término general para referirse a una línea en un espacio de alta dimensionalidad es el *hiperplano* y el *hiperplano de separación* es aquel que permite la división de las clases de datos. Ahora bien, el *hiperplano óptimo* es el que maximiza la distancia entre los datos de clases diferentes.

En ocasiones, los datos no pueden ser separados con una simple línea recta, por lo que el algoritmo de la máquina de vectores soporte debe ser modificado para permitir el manejo de *márgenes suaves*, es decir, tolerar que algunos puntos de datos se localicen en el lado erróneo del hiperplano. La *función kernel* proyecta los datos a un espacio de mayor dimensionalidad, conocido como espacio de características, en búsqueda del hiperplano óptimo suponiendo que en este espacio, los datos son linealmente separables.

Capítulo III

Kernels para grafos

La necesidad por tener datos áltamente estructurados surge por la facilidad que permiten a la hora de ser clasificados por algoritmos empleados en el aprendizaje de máquina. Cuando esto sucede, las funciones de comparación son relativamente sencillas y es por eso que es fácil distinguir las diferencias entre los datos de entrada para ser clasificados.

Sin duda, esta situación es la más deseada cuando se piensa en implementar algún algoritmo de este tipo, sin embargo en ocasiones, la naturaleza de los objetos es irregular y obliga a la implementación de algoritmos de extracción de características o de representación, además del que se encarga de la propia clasificación.

Mientras que existen datos que pueden ser representados como cadenas, vectores o inclusive números reales, existen otros objetos que, dado su complejidad, la mejor manera de representarlos es en forma de grafos para poder expresar la relación entre sus componentes.

Para estos casos, los métodos kernels que trabajan con datos en forma de vectores no son compatibles directamente. Es por esto que surge la necesidad de los kernels para grafos que extraigan, en la medida de lo posible, toda la información expresada en el grafo representando el objeto a analizar.

Existen distintos acercamientos en la literatura ofreciendo una solución a este problema. En este trabajo nos enfocaremos en cinco distintos ya que son los que se consideran más relevantes en la literatura actualmente.

Para esto, primero es necesario establecer las bases teóricas necesarias para su comprensión. A continuación se presenta un conjunto de definiciones dentro del campo de la teoría de grafos así como algunas convenciones que serán usadas con frecuencia en el resto del trabajo. En su mayoría, las definiciones son extraidas de (Vishwanathan *et al.*, 2010).

III.1 Teoría de grafos

Un *grafo* $G = (V, E)$ consiste en un conjunto de n vértices $V = \{v_1, v_2, \cdots, v_n\}$ y m aristas $E = \{e_1, e_2, \cdots, e_m\} \subseteq V \times V$. Se dice que un vértice v_i es vecino de otro v_j si los conecta una arista, esto es, si $(v_i, v_j) \in E$ (también se puede denotar como $v_i \sim v_j$). Se dice que un grafo es *no dirigido* si $(v_i, v_j) \in E \iff (v_j, v_i) \in E$.

Un *grafo con pesos* es aquel que para cada arista $(v_i, v_j) \in E$ existe un peso asociado $w_{ij} > 0$. Si v_i y v_j no están conectados, entonces $w_{ij} = 0$ y en un grafo no dirigido $w_{ij} = w_{ji}$.

Una *caminata* de tamaño k en G es una secuencia de índices $i_0, i_1, \cdots, i_k$ tal que $(v_{i_{r-1}}, v_{i_r}) \in E$, para todo $1 \leq r \leq k$.

Un *camino* de tamaño k en G, al igual que una caminata, es un conjunto de índices $i_0, i_1, \cdots, i_k$ tal que $(v_{i_{r-1}}, v_{i_r}) \in E$ para todo $1 \leq r \leq k$ pero además $v_i \neq v_j$ para todo $v_i, v_j \in V$. Un *camino de aristas* es un camino el cual puede tener vértices repetidos pero no contiene aristas repetidas, esto es, $e_i \neq e_j$ para todo $e_i, e_j \in E$. En este trabajo se referirá a los *caminos de aristas* solo como *caminos* por ser los que se requieren a lo largo del mismo.

Un grafo *conectado* es aquel que para cualquier par de vértices v_i y v_j existe un *camino* entre ellos. Se dice que un grafo es *fuértemente conectado* si para cualquier par de vértices v_i y v_j existe una arista $(v_i, v_j) \in E$.

Un *árbol* (Figura 7 a) es un grafo conectado, acíclico y no dirigido. Un *árbol enraizado* cuenta con un vértice especial llamado *raíz*. Considerando un *camino* desde la raíz v_r hasta un vértice v_i, si tenemos una arista (v_{j-1}, v_j) en dicho camino, entonces v_{j-1} es el *padre* de v_j y v_j es el *hijo* de v_{j-1}. La raíz v_r es el único vértice del árbol sin *padre* (Cormen *et al.*, 2001).

Se le llama *hoja* al vértice v_i cuando éste no tiene hijos. La *altura* h de un árbol es el la longitud en número aristas que se deben de recorrer entre la raíz y su hoja más lejana. Los árboles pueden ser clasificados por el número máximo de hijos que cada vértice puede

tener, por ejemplo, un *árbol binario* es aquel que solo admite tener a los más 2 hijos en cada vértice.

Un *subárbol* es aquel árbol T' en el que sus vértices y aristas forman un subconjunto de los vértices y aristas de un árbol T dado (Weisstein, 2011b).

Un *árbol de esparcimiento* de un grafo es un subconjunto de $n-1$ aristas que forman un árbol. Cuando además, para un grafo con pesos en sus aristas, éstas tienen un peso total mínimo, entonces se le llama *árbol de esparcimiento mínimo*.

Una de las características más importantes en el uso de los árboles binarios para su aplicación en problemas como la búsqueda de información es la manera sistemática de recorrerlos de tal manera que se visite cada vértice solo una vez para propósitos de lectura o actualización. Los 3 recorridos más populares son los recorridos *pre-orden*, *post-orden* e *inorden*.

El recorrido *pre-orden* es aquel en el que en el que primeramente se visita la raíz y posteriormente se recorre de la misma manera el subárbol izquierdo seguido por el derecho. Por otro lado, el recorrido *post-orden* es aquel en el que se recorre primeramente el subárbol izquierdo, seguido del subárbol derecho y por último se visita la raíz. Finalmente, en el recorrido *inorden* se recorre el subárbol izquierdo, después se visita la raíz y por último se recorre el subárbol derecho.

Una *estrella* (Figura 7 b) S_n es un árbol de n vértices en el cual uno de los vértices está conectado a todos los demás. Se puede ver a una estrella como un árbol de altura $h = 1$ con $n - 1$ hojas.

Un *anillo* (Figura 7 c) es un grafo que forma un camino el cual empieza y termina en el mismo vértice. Esto es, un anillo G de tamaño n es un conjunto de vértices $v_{i_1}, v_{i_2}, \cdots, v_{i_n}$ tal que $(v_{i_{r-1}}, v_{i_r}) \in E$, para todo $1 \leq r \leq k$ y además $v_{i_1} = v_{i_n}$.

El *diámetro* de un grafo es el más largo de los caminos más cortos entre cualquier par de vértices v_i y v_j de un grafo. Esto es $max_{i,j}d(i,j)$ entre cualquier par de vértices i y j del grafo en donde $d(i,j)$ es la distancia más corta entre el vértice i y el vértice j (Weisstein,

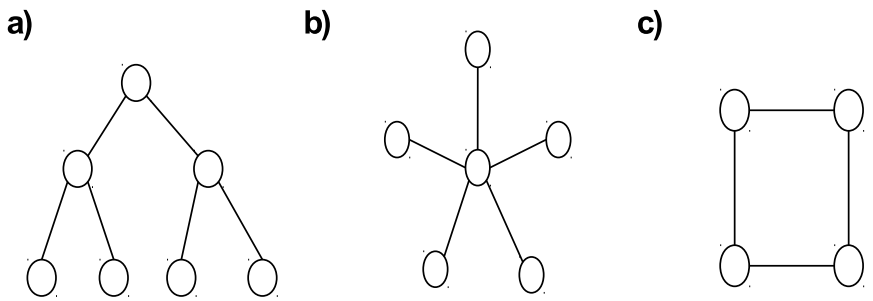


Figura 7. Ejemplos de distintas topologías de grafos. a) *Anillo* tamaño $n = 4$. b) *Estrella* tamaño $n = 6$ c) árbol binario de altura $h = 3$.

2011a).

Si se toma un grafo conectado con un diámetro grande y se le agregan un número relativamente chico de aristas de manera aleatoria, el diámetro de dicho grafo disminuirá drásticamente. A esto se le conoce como el fenómeno de la *red de mundo pequeño* (Pegg, 2011)(ver Figura 8). Este fenómeno se da en muchos escenarios y quizá el más popular es el ejemplo de las redes sociales en el mundo en donde se dice que cualquier par de personas están relacionadas entre si a través de aproximadamente 6 personas.

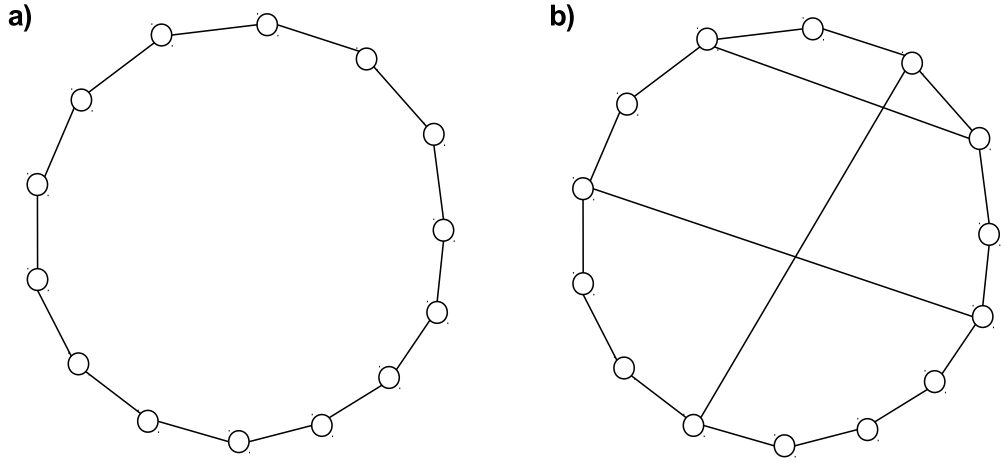


Figura 8. Ejemplo de una red de mundo pequeño. a) *Anillo* tamaño $n = 14$ el díametro es $D = 7$. b) Al agregar 3 aristas de manera aleatoria al mismo grafo, el diámetro se redujo a $D = 5$

Para grafos con pesos se define su *matriz de adyacencia* como la matriz $\widetilde{A}$ de $n \times n$ con $\widetilde{A}_{ij} = 1$ si $v_i \sim v_j$ y 0 de lo contrario. Para grafos con pesos, $\widetilde{A}_{ij} = w_{ji}$. Para grafos no

dirigidos, la matriz de adyacencias $\widetilde{A}$ es simétrica. Los valores de la diagonal de $\widetilde{A}$ siempre son ceros (si no hay autociclos).

La *matriz de adyacencias normalizada* $A = \widetilde{A}D^{-1}$, tiene la propiedad de que cada una de sus filas suma 1, es por esto que puede servir como una matriz de transición de procesos estocásticos. En este caso D es una matriz diagonal de los grados de los vértices, esto es, $D_{ij} = d_i = \sum_j \widetilde{A}_{ij}$.

Una *caminata aleatoria* en G es un proceso que genera secuencias de vértices $v_{i_1}, v_{i_2}, \cdots$ de acuerdo a $\mathbb{P}(i_{k+1}|i_1, \cdots, i_k) = A_{i_{k+1},i_k}$, esto es, la probabilidad de pasar a $v_{i_{k+1}}$ estando en v_{i_k} es proporcional al peso de la arista $(v_{i_k}, v_{i_{k+1}})$. La t-ésima potencia de A describe caminatas de longitud t, esto es, $(A^t)_{ij}$ es la probabilidad de una transición del vértice v_j al vértice v_i a través de una caminata aleatoria de longitud t. Si p_0 es la distribución de probabilidad inicial sobre vértices, entonces la distribución de probabilidad p_t describiendo la ubicación de la caminata en el tiempo t es $p_t = A^t p_0$. El j-ésimo componente de p_t denota la probabilidad de terminar una caminata de longitud t en el vértice v_j.

No es necesario que la caminata aleatoria continue indefinidamente, para evitar esto, se asocia a cada vértice v_{i_k} en el grafo una probabilidad de paro q_{i_k}. El *kernel de caminatas aleatorias para grafos* generalizado usa la probabilidad de detenerse después de t pasos, dado por $q^T p_t$. Como p_0, el vector q de probabilidades de paro es en donde se puede integrar algún conocimiento previo en el diseño del kernel. Ya que la suma de la distribución de probabilidad p_t es 1, un vector uniforme q resultaría en la misma probabilidad de paro para todo p_t, resultando en un kernel que es invariante con respecto a la estructura que fue diseñado a medir.

Sea $\mathcal{L}$ un conjunto de etiquetas que incluye la etiqueta especial ζ. Cada grafo G con aristas etiquetadas está asociado a una matriz de etiquetas $X \in \mathcal{L}^{n\times n}$ en la cual X_{ij} es la etiqueta de la arista (v_i, v_j) y $X_{ij} = \zeta$ si $(v_i, v_j) \notin E$. Sea $\mathcal{H}$ el *RKHS* inducido por un kernel positivo semi-definido $\kappa : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$, y sea $\phi : \mathcal{L} \to \mathcal{H}$ el mapa de características correspondiente, que se supone mapea ζ al elemento zero en $\mathcal{H}$. Se usa $\Phi(X)$ para denotar

el mapa de características de G.

Dos grafos $G = (V, E)$ y $G' = (V', E')$ son *Isomorfos* (denotado por $G \cong G'$) si existe un mapeo biyectivo $g : V \to V'$ (llamado función de isomorfismo) tal que $(v_i, v_j) \in E$ si y solo si $(g(v_i), g(v_j)) \in E'$.

Dado dos grafos $G(V, E)$ y $G'(V', E')$, su *producto directo* G_x es un grafo con un conjunto de vértices

$$V_x = \{(v_i, v'_r) : v_i \in V, v'_r \in V'\},$$

y el conjunto de aristas

$$E_x = \{((v_i, v'_r)(v_j, v'_s)) : (v_i, v_j) \in E \wedge (v'_r, v'_s) \in E'\}.$$

La Figura 9 ilustra el producto directo entre grafos. El grafo G_x es un grafo sobre pares de vértices de G y G' y dos vértices en G_x son vecinos si y solo si los dos vértices correspondientes en G y G' son ambos vecinos. Si $\widetilde{A}$ y $\widetilde{A}'$ son las matrices de adyacencias de G y G' respectivamente, entonces la matriz de adyacencia de G_x es $\widetilde{A}_x = \widetilde{A} \otimes \widetilde{A}'$. También, para las matrices de adyacencias normalizadas, $A_x = A \otimes A'$.

Definición 2. Dadas dos matrices reales $A \in \mathbb{R}^{n \times m}$ y $B \in \mathbb{R}^{p \times q}$, el producto de Kronecker $A \otimes B \in \mathbb{R}^{np \times mq}$ y el operador de apilamiento de columnas $vec(A) \in \mathbb{R}^{n \times m}$, están dados por

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \cdots & A_{nm}B \end{bmatrix}, vec(A) = \begin{bmatrix} A_{*1} \\ \vdots \\ A_{*m} \end{bmatrix},$$

donde A_{*j} se refiere a la columna $j - ésima$ de A.

El producto de Kronecker y el operador *vec* se relacionan por la siguiente propiedad (Bernstein (2005), Proposición 7.1.9):

$$vec(ABC) = (C^T \otimes A)vec(B). \tag{12}$$

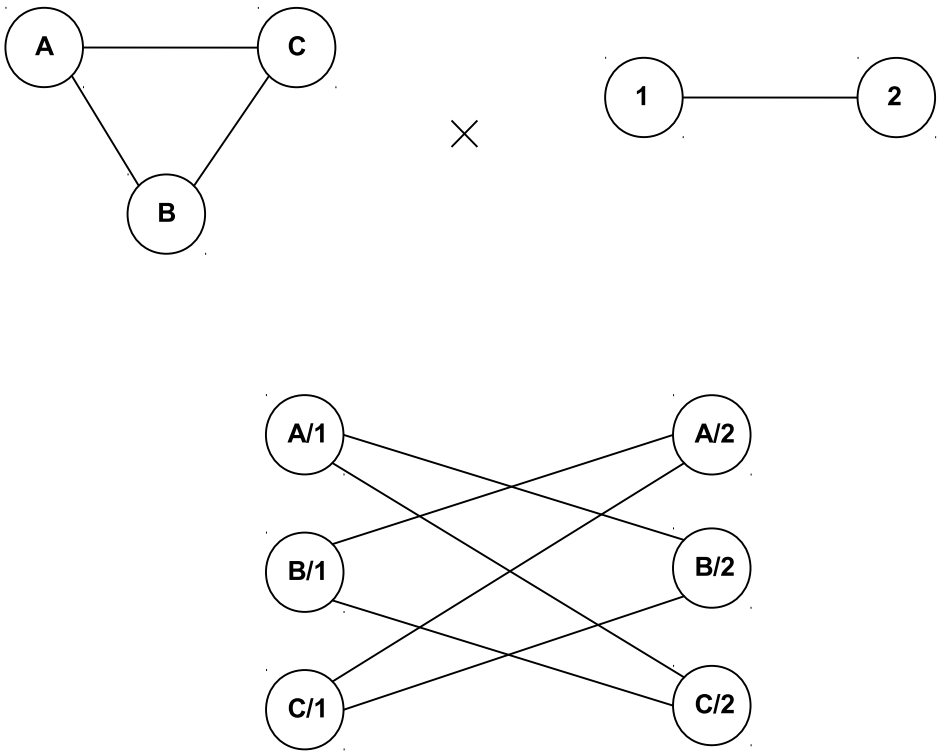


Figura 9. Dos grafos (arriba) y su grafo de producto directo (abajo).

Estos conceptos pueden ser extendidos al espacio de Hilbert $\mathcal{H}$ correspondiete a los kernels continuos, simétricos y positivos definidos. Este espacio es llamado espacio de Hilbert de reproducción del kernel (RKHS por sus siglas en inglés).

La siguiente es una extensión de (12) al RKHS presentada en Vishwanathan *et al.* (2010) (Lema 12):

Si $A \in \mathcal{X}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$ y $C \in \mathcal{X}^{p\times q}$, entonces

$$vec(\Phi(A)B\Phi(C)) = (\Phi(C)^T \otimes \Phi(A))vec(B) \in \mathbb{R}^{nq\times 1}. \tag{13}$$

III.2 Kernel basado en caminatas aleatorias

Los kernels para grafos basados en caminatas aleatorias se basan en una idea simple: dado un par de grafos, realizar una caminata aleatoria en ambos y contar el número de caminatas iguales. Vishwanathan *et al.* (2010) hacen una evaluación de los distintos acercamientos a

los kernels para grafos basados en caminatas aleatorias y muestran que este es un concepto subyacente tanto en los kernels para grafos basados en caminatas aleatorias como en los kernels marginales.

Realizar una caminata aleatoria en un grafo del *producto directo* es equivalente a realizar una caminata aleatoria en G y G' simultaneamente (Imrich y Klavzar, 2000). Si p y p' es la distribución de probabilidad de inicio sobre los vértices de G y G', entonces la distribución de probabilidad inicial en el grafo de producto directo es $p_x = p \otimes p'$. De la misma manera, si q y q' es la distribución de probabilidad de paro sobre los vértices de G y G', entonces la distribución de probabilidad de paro en el grafo de producto directo es $q_x = q \otimes q'$.

Sean n y n' el número de vértices de G y G' respectivamente. Si los grafos G y G' son de aristas etiquetadas, se puede asociar una matriz de peso $W_x \in \mathbb{R}^{nn' \times nn'}$ con G_x usando la extensión del producto de Kronecker (Definición 2) al RKHS:

$$W_x = \Phi(X) \otimes \Phi(X'). \tag{14}$$

Como consecuencia de la definición de $\Phi(X)$ y $\Phi(X')$, los valores de W_x son distintos a cero solo si existe la arista correspondiente en el grafo de producto directo. Si hacemos que $(H) = \mathbb{R}, \Phi(X) = \widetilde{A}$, y $\Phi(X') = \widetilde{A}'$, entonces (14) se reduce a $\widetilde{A}_x$, la matriz de adyacencias del grafo de producto directo. Se puede incorporar la normalización haciendo $\Phi(X_{ij}) = 1/d_i$ si $(v_i, v_j) \in E$, y cero de lo contrario. De esta manera, $\Phi(X) = A$ y $\Phi(X') = A'$, y como consecuencia $W_x = A_x$ (Vishwanathan *et al.*, 2010).

Si las etiquetas de las aristas forman un conjunto finito, sin pérdida de generalidad $\{1, 2, \cdots, d\}$, podemos hacer $\mathcal{H}$ un $\mathbb{R}^d$ dotado con el producto interno habitual. Sea $\mathbf{e}_i$ un vector con todos sus elementos igual a 0 y solo el i-ésimo igual a 1 y $\mathbf{0}$ un vector con todos sus elementos iguales a 0, para cada arista $(v_i, v_j) \in E$ asignamos a $\phi(X_{ij}) = \mathbf{e}_l/d_i$ si la arista (v_i, v_j) tiene la etiqueta l; el resto de los valores de $\Phi(X)$ son $\mathbf{0}$. Por lo tanto la matriz de pesos (14) tiene un elemento distinto a cero si y solo si existe una arista en el grafo de producto directo y sus aristas correspondientes en G y G' tienen la misma etiqueta. Sea ${}^lA$

la matriz de adyacencias normalizada del grafo filtrado por la etiqueta l, esto es ${}^{l}A_{ij} = l$, y cero de lo contrario. La matriz de pesos del grafo de producto directo puede ser escrito como

$$W_x = \sum_{l=1}^{d} {}^{l}A \otimes {}^{l}A'. \tag{15}$$

III.2.1 Definición del kernel

Realizar una caminata aleatoria sobre el grafo G_x es equivalente a realizar una caminata aleatoria simultanea en los grafos G y G' según Imrich y Klavzar (2000). Por lo tanto, el $((i-1)n'+r,(j-1)n'+s)$-ésimo elemento de A_x^k representa la probabilidad de dos caminatas aleatorias simultaneas, una desde v_j hastsa v_i y la otra desde v'_s hasta v'_r en G y G', respectivamente. Los valores de W_x (14) representan la similitud entre aristas, de tal forma que el elemento $((i-1)n'+r,(j-1)n'+s)$ de W_x^k representa la similitud entre caminatas simultaneas de longitud k en G y G' medida por la función de kernel κ. Dadas las distribuciones de probabilidad de inicio y paro p_x y q_x respectivamente, se puede calcular $q_x^T W_x^k p_x$ que es la similitud esperada entre dos caminatas aleatorias de longitud k en G y G' (Vishwanathan *et al.*, 2010).

Con todo esto, la idea más natural para definir el kernel que calcula la similitud entre dos grafos G y G' sería sumar $q_x^T W_x^k p_x$ para todo k. No obstante, existe el riesgo de que la suma no converja, lo que dejaría sin definir al valor del kernel. Para solventar el problema, se introduce un coeficiente no negativo $\mu(k)$ escogido apropiadamente y se define el kernel entre G y G' como

$$k(G,G') = \sum_{k=0}^{\infty} \mu(k) q_x^T W_x^k p_x. \tag{16}$$

Esta estructura ofrece mucha flexibilidad a los diseñadores de kernels para su aplicación a distintos problemas ya que se tienen tres parámetros a su disposición: por un lado, $\mu(k)$ se puede modificar para así asignarle una importancia a la longitud de la caminata pues, como se mencionó, es un factor de "degradación" para evitar que las caminatas se extiendan indefinidamente; por otro lado, si se conocen las probabilidades de inicio y paro

en una aplicación en particular, se puede incorporar este conocimiento en el kernel; por último, se puede incorporar medidas de similitud apropiadas entre aristas a través de la matriz de pesos W_x. A pesar de dicha flexibilidad, Vishwanathan *et al.* (2010), haciendo uso de (12), muestran que este kernel es positivo definido y -como se mostrará más adelante- que se puede calcular eficientemente aprovechando la estructura de W_x.

III.2.2 Casos especiales

Vishwanathan *et al.* (2010) adaptan su kernel para grafos basado en caminatas aleatorias generalizado a dos kernels para grafos resultantes de otros dos trabajos. Por un lado, el trabajo de Kashima *et al.* (2003) llamado kernel marginal y por otro lado, otro enfoque basado en caminatas aleatorias hecho por Gärtner *et al.* (2003).

III.2.2.1 Kernel Marginal

Kashima *et al.* (2003) definen un kernel para grafos etiquetados utilizando las secuencias de etiquetas producidas por caminatas aleatorias de tal manera que una caminata de longitud t en un grafo G es una secuencia de índices $i_1, i_2, \cdots, i_{t+1}$ de tal manera que $(v_{i_k}, v_{i_{k+1}}) \in E$ para todo $1 \leq k \leq t$. En este caso, la secuencia de etiquetas $h = h_1, h_2, \cdots, h_t$ asociada a una caminata es la secuencia de etiquetas de las aristas que se recorren en la propia caminata. Sea P la matriz de probabilidad, en donde P_{ij} denota la probabilidad de la transición de un vértice v_i al vértice v_j de tal manera que P pudiera ser la matriz de adyacencias normalizada de G. Además, sean p y q las probabilidades de inicio y paro respectivamente. Entonces se puede calcular la probabilidad de la caminata $i_1, i_2, \cdots, i_{t+1}$ y por lo tanto la secuencia de etiquetas h asociada a esta caminata de la siguiente manera:

$$p(h|G) = q_{i_{t+1}} \prod_{j=1}^{t} P_{i_j, i_{j+1}} p_{i_1}. \tag{17}$$

Sea $\hat{\phi}$ un mapa de características de las etiquetas en aristas, se define un kernel entre

las secuencias de etiquetas de tamaño t

$$\kappa(h,h') = \prod_{i=1}^{t} \kappa(h_i, h'_i) = \prod_{i=1}^{t} \left\langle \hat{\phi}(h_i), \hat{\phi}(h'_i) \right\rangle \tag{18}$$

si las secuencias h y h' tienen la misma longitud y cero en caso contrario. Usando (17) y (18) se puede definir un kernel entre grafos mediante la marginalización:

$$k(G,G') = \sum_{h} \sum_{h'} \kappa(h,h') p(h|G) p(h|G'). \tag{19}$$

Kashima *et al.* (2004) (Ecuación 1.19) muestran que este kernel (19) puede ser escrito de la siguiente manera:

$$k(G,G') = q_x^T (\mathbf{I} - T_x)^{-1} p_x, \tag{20}$$

en donde $T_x = [\text{vec}(P)\text{vec}(P')^T] \odot [\hat{\Phi}(X) \otimes \hat{\Phi}(X')]$. X y X' son las matrices de G y G', respectivamente y $\hat{\Phi}$ las matrices de características correspondientes.

Este kernel se puede obtener como un caso especial del marco de trabajo propuesto por Vishwanathan *et al.* (2010), para esto, se hace $\mu(k) = \lambda^k$ para algún $\lambda > 0$. Entonces se puede escribir:

$$k(G,G') = \sum_{k=0}^{\infty} \lambda^k q_x^T W_x^k p_x = q_x^T (\mathbf{I} - \lambda W_x)^{-1} p_x. \tag{21}$$

Para recuperar el kernel para grafo marginal se hace $\lambda = 1$ y se define $\Phi(X_{ij}) = P_{ij}\hat{\Phi}(X_{ij})$ de tal manera que $W_x = T_x$ y así obteniendo (20).

III.2.2.2 Kernel de Gärtner

Gärtner *et al.* (2003) también realizan caminatas aleatorias en un par de grafos dados pero en este caso cuentan el número de caminatas coincidentes. Su kernel se define como (Gärtner *et al.*, 2003) (Def. 6):

$$k(G,G') = \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \lambda_k [\widetilde{A}_x^k]_{ij}. \tag{22}$$

Este kernel se puede obtener en el marco de trabajo utilizado por Vishwanathan *et al.* (2010) asignando $\mu(k) = \lambda_k$ y suponiendo distribuciones uniformes de inicio y paro sobre

los vértices de G y G', esto es, $p_i = q_i = 1/n$ y $p'_i = q'_i = 1/n'$ y sea $\Phi(X) = \widetilde{A}$ y $\Phi(X)' = \widetilde{A}'$. Por consiguiente, $p_x = q_x = \mathbf{e}/(nn')$, y $W_x = \widetilde{A}_x$ (la matriz de adyacencias no normalizada de la matriz de producto). Esto permite reescribir (16) para obtener:

$$k(G, G') = \sum_{k=0}^{\infty} \mu(k) q_x^T W_x^k p_x = \frac{1}{n^2 n'^2} \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \lambda_k [\widetilde{A}_x^k]_{ij}, \tag{23}$$

que recupera (23) dentro de un factor constante. Este kernel también puede ser extendido a grafos con un conjunto finito de etiquetas reemplazando $\widetilde{A}_x$ en (23) con una sumatoria $\widetilde{W_x}$ de matrices de adyacencias no normalizadas filtradas por etiquetas, análoga a (15). Gärtner *et al.* (2003) discuten dos casos de interés: primero, su kernel *geométrico* emplea un factor de descomposición λ fijo para reducir la contribución de caminatas largas al kernel haciendo $\lambda_k = \lambda^k$ como en (21). La selección de λ es crítica ya que debe ser lo suficientemente chica para que la sumatoria en (22) converja. Segundo, el kernel *exponencial* se define como:

$$k(G, G') = \sum_{i=1}^{n} \sum_{j=1}^{n'} [e^{\lambda \widetilde{A}_x}]_{ij} = \mathbf{e}^T e^{\lambda \widetilde{A}_x} \mathbf{e}, \tag{24}$$

usando la exponenciación de matrices, en donde $\mathbf{e}$ denota un vector formado solo de unos. Esto se obtiene en el marco de trabajo de Vishwanathan *et al.* (2010) haciendo $\lambda_k = \lambda^k / k!$, de tal modo que la sumatoria de la derecha en (23) se convierte en la expansión de series $e^{\lambda \widetilde{A}_x}$. En este caso, el kernel de Gärtner *et al.* (2003) difiere de la definición de Vishwanathan *et al.* (2010) (16) en que no se modelan las probabilidades de inicio y de paro y además utiliza la matriz de adyacencias no normalizada en vez de una matriz de pesos más general (14) que permite la normalización y kernels en aristas arbitrarios.

III.2.3 Cómputo eficiente

El problema de este kernel es que el tiempo requerido de computar es de $O(n^6)$ ya que es equivalente a invertir una matriz de tamaño $n \times n$ dados dos grafos G y G' con n vértices cada uno.

Vishwanathan *et al.* (2010) desarrollaron métodos que se basan en la ecuación de Sylvester, gradiente conjugado, iteración de punto fijo, descomposición espectral y una aproximación al producto de Kronecker que aceleran en gran medida el cómputo de estos kernels.

La aplicabilidad y complejidad en tiempo de un método dependerá si el grafo es etiquetado posiblemente con pesos en las aristas ($W_x = A_x$), tiene etiquetas de un conjunto finito (15) o, un conjunto infinito de etiquetas (14).

A continuación se describen brevemente cada uno de los métodos utilizados:

Tabla II. Complejidad en los peores casos (Notación en $O(\cdot)$) de los métodos para una matriz de $m \times m$ del kernel para grafo de caminatas aleatorias.

Densidad de grafo	Denso				Disperso
etiq. en aristas Método $W_x =$	sin/escalar $A \otimes A'$	conj. finito (15)	dim. finita kernel (14)	dim. ∞ kernel (14)	cualquier
Ecuación de Sylvestre	m^2n^3	desconocido	-		-
Gradiente conjugado	m^2rn^3	m^2rdn^3		m^2rn^4	m^2rn^2
Iteración de punto fijo	m^2kn^3	m^2kdn^3		m^2kn^4	m^2kn^2
Descomposición espectral	$(m+n)mn^2$	m^2n^6			-
Producto de Kronecker más cercano	1	$m^2k'dn^2$	$m^2k'n^4$		m^2kn^2

donde m = número de grafos, n = número de vértices, d = tamaño del conjunto de etiquetas o la dimensionalidad del mapa de características en donde corresponda, r = rango efectivo de W_x, k = número de iteraciones de punto fijo (36) y k' = número de iteraciones de potencia (40) (Tabla extraida de Vishwanathan *et al.* (2010), 1210).

III.2.3.1 Ecuación de Sylvester

La siguiente ecuación es comunmente llamada ecuación de Sylvester o de Lyapunov:

$$M = SMT + M_0. \tag{25}$$

Donde $S, T, M_0 \in \mathbb{R}^{n \times n}$ son dados y se necesita resolver $M \in \mathbb{R}^{n \times n}$. Estas ecuaciones pueden ser resueltas en $O(n^3)$ (Gardiner *et al.*, 1992) con código disponible como lo es el método *dlyap* de *Matlab*. Resolver la ecuación de Sylvester generalizada

$$M = \sum_{i=1}^{d} S_i M T_i + M_0 \tag{26}$$

requiere computar factorizaciones de Schur generalizadas de d matrices simétricas simultaneamente que, aunque más costoso, puede ser resuelto eficientemente. La complejidad computacional de esta factorización generalizada es desconocida en la actualidad (Vishwanathan *et al.*, 2010).

Vishwanathan *et al.* (2010) mostraron que para grafos con etiquetas discretas en las aristas, para la cual su matriz de pesos W_x puede ser escrita como (14), el problema de computar el kernel para grafos (21) puede ser reducido a resolver la siguiente ecuación de Sylvestre generalizada:

$$M = \sum_{i=1}^{d} \lambda^i A' M^i A^T + M_0, \tag{27}$$

en donde $\text{vec}(M_0) = p_x$. Se empieza a "aplanar" la ecuación (27)

$$\text{vec}(M_0) = \lambda \sum_{i=1}^{d} \text{vec}({}^iA' M^i A^T) + p_x. \tag{28}$$

Usando (13), que es una extensión de (12) a un RKHS, se puede reescribir (28) como

$$(\mathbf{I} - \lambda \sum_{i=1}^{d} {}^iA \otimes {}^iA')\text{vec}(M) = p_x, \tag{29}$$

usando (15), y despejando $\text{vec}(M)$ en (29):

$$\text{vec}(M) = (\mathbf{I} - \lambda W_x)^{-1} p_x. \tag{30}$$

Después, multiplicando ambos lados de (30) por q_x^T tenemos:

$$q_x^T \text{vec}(M) = q_x^T (\mathbf{I} - \lambda W_x)^{-1} p_x. \tag{31}$$

La parte derecha de (31) es el kernel para grafos (21). Dada la solución M de la ecuación de Sylvestre (27), el kernel para grafos puede ser obtenido como $q_x^T \text{vec}(M)$ en $O(n^2)$. De la misma manera, para grafo grafos no etiquetados simplemente se hace $d = 1$ para poder aplicar el mismo argumento y con esto convertir a (27) en una ecuación de Sylvestre simple. Esto supone una mejora considerable al método directo para computar el kernel para grafos basado en caminatas aleatorias ($O(n^6)$) ya que tiene un tiempo de ejecución de $O(n^3)$.

La sofisticación de los métodos disponibles para resolver la ecuación de Sylvester limitan su aplicabilidad; además de que regularmente se encuentran solo como rutinas de librerías sin dar a conocer el código. No obstante, se puede utilizar un método para resolver la ecuación de Sylvester simple (25) para computar kernels entre grafos etiquetados (en aristas) eficientemente empleando la aproximación del producto de Kronecker más cercano (III.2.3.5).

III.2.3.2 Gradiente conjugado

Dada una matriz M y un vector b, el método del gradiente conjugado (GC) resuelve la ecuación $Mx = b$ eficientemente (Nocedal y Wright, 1999). Estos métodos están diseñados para su uso con matrices simétricas y positivas semi-definidas pero también pueden ser utilizadas para resolver otros sistemas lineales eficientemente. Son particularmente eficientes si la matriz es deficiente en su rango o tiene un *rango efectivo* pequeño, esto es, el número de eigen valores distintos. Además, si el cálculo de productos *matriz-vector* es computacionalmente barato, el tiempo para resolver el GC puede ser mejorado considerablemente (por ejemplo cuando la matriz M es dispersa). Suponiendo que computar Mv para un vector v arbitrario requiere un tiempo de $O(m)$ y el *rango efectivo* de M es r, entonces el método GC toma $O(r)$ iteraciones y por lo tanto requiere $O(mr)$ para resolver $Mx = b$.

El kernel para grafos (21) puede ser computado por un procedimiento de dos pasos: Primero se resuelve el sistema lineal

$$(\mathbf{I} - \lambda W_x)x = p^x, \tag{32}$$

para x, despues se computa $q_x^T x$. Ahora el enfoque es el resolver eficientemente (32) usando el método de GC. Recordemos que si G y G' contienen n vértices, entonces W_x es una matriz de $n^2 \times n^2$. Dentro del algoritmo de GC, multiplicar W por un vector y requiere $O(n^4)$ operaciones, pero existe una forma de usar la extensión de (12) a un RKHS (ecuación 13) para mejorar el tiempo de ejecución. Se introduce la matriz $Y \in \mathbb{R}^{n \times n}$ con $y = \mathrm{vec}(Y)$,

y recordando que $W_x = \Phi(X) \otimes \Phi(X')$, por (13) se puede escribir

$$W_x y = (\Phi(X) \otimes \Phi(X'))\text{vec}(Y) = \text{vec}(\Phi(X')Y\Phi(X)^T). \tag{33}$$

Si $\phi(\cdot) \in \mathbb{R}^d$ entonces este producto matriz-vector puede ser computado en un tiempo de $O(dn^3)$. Si $\Phi(X)$ y $\Phi(X')$ son dispersas, entonces $\Phi(X')Y\Phi(X)^T$ puede ser computado en aun menor tiempo, esto es, si hay $O(n)$ elementos distintos a ξ en $\Phi(X)$ y $\Phi(X')$, calcular (33) toma un tiempo de $O(n^2)$.

III.2.3.3 Iteración de punto fijo

Otro de los métodos con los cuales Vishwanathan *et al.* (2010) calculan el kernel de caminatas aleatorias es el de *iteración de punto fijo.*

El método de iteración de punto fijo comienza reescribiendo (32) como

$$x = p_x + \lambda W_x x. \tag{34}$$

Resolver x es equivalente a encontrar un punto fijo de (34) tomado como una iteración. Ahora, si se toma a x_t como el valor de x en la iteración t, se asigna $x_0 = p_x$ y se computa

$$x^{t+1} = p_x + \lambda W_x x_t, \tag{35}$$

repetidamente hasta $||x_{t+1} - x_t|| < \varepsilon$, en donde $||\cdot||$ denota la norma Euclidiana y ε una tolerancia predefinida. Esto está garatizado a converger siempre y cuando si todos los eigenvalores de λW_x se encuentran dentro del disco unitario, lo que se puede asegurar asignando $\lambda < |\xi_1|^{-1}$, en donde ξ_1 es el eigenvalor de W_x de mayor magnitud. Asumiendo que cada iteración de (35) contrae x al punto fijo dentro de un factor de $\lambda\xi_1$, se converge hasta estar a un factor ε de del punto fijo en k iteraciones, en donde

$$k = O\left(\frac{\ln \varepsilon}{\ln \lambda + \ln |\xi_1|}\right). \tag{36}$$

Ya que cada iteración de (35) involucra calcular del producto matriz-vector $W_x x_t$, todas las mejoras al tiempo de cálculo de productos matriz-vector vistos en la sección (III.2.3.2) son

aplicables en este caso también. El hecho que W_x es una suma de productos de Kronecker es particularmente importante ya que se explota para reducir la complejidad en tiempo del peor caso a $O(dn^3)$ por iteración en experimentos realizados por Vishwanathan *et al.* (2010) a comparación de los resultados obtenidos por Kashima *et al.* (2004) que calcularon los productos matriz-vector explícitamente.

III.2.3.4 Descomposición espectral

En las secciones (III.2.3.2) y (III.2.3.3), Vishwanathan *et al.* (2010) introdujeron métodos que son eficientes para grafos etiquetados y no etiquetados específicamente calculando el kernel geométrico (21) asumiendo que $\mu(k) = \lambda^k$. En esta sección se trata un método basado en la descomposición espectral que permite computar el kernel basado en caminatas aleatorias general (16) para cualquier selección de $\mu(k)$ siempre y cuando converja, pero que solo es eficiente para grafos no etiquetados.

Sea $W_x = P_x D_x P_x^{-1}$ donde las columnas de P_x son los eigenvectores y D_x es una matriz diagonal de los eigenvalores correspondientes de W_x respectivamente, a esto se le denomina la descomposición espectral de W_x. El kernel de caminata aleatoria (16) puede ser escrito como

$$k(G, G') = \sum_{k=0}^{\infty} \mu(k) q_x^T (P_x D_x P_x^{-1})^k p_x = q_x^T P_x \left(\sum_{k=0}^{\infty} \mu(k) D_x^k \right) P_x^{-1} p_x. \tag{37}$$

Esto simplifica las cosas ya que (38) solo toma potencias ponderadas de una matriz diagonal, lo que se desompone en potencias escalares de sus elementos. Para obtener un kernel para grafos implementable, basta con emplear una serie de potencias la cual se sepa que converja a una función no linear dada. Por ejemplo, el kernel geométrico (21) usa el hecho de que $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ y haciendo $\mu(k) = \lambda^k$ en (37) se obtiene

$$k(G, G') = q_x^T P_x (\mathbf{I} - \lambda D_x)^{-1} P_x^{-1} p_x. \tag{38}$$

La diferencia más importante con (21) es que la inversión en (38) es sobre una matriz diagonal, lo que es trivial de computar. Si se hace $\mu(k) = \frac{\lambda^k}{k!}$ en (37) da como resultado el

kernel exponencial (24) por medio de la descomposición espectral:

$$k(G, G') = q_x^T P_x e^{\lambda D_x} P_x^{-1} p_x, \tag{39}$$

ya que $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$. De nuevo, la exponenciación de la matriz es trivial de calcular (ya que λD_x es diagonal) elevando cada uno de sus elementos, a diferencia de (24).

De esta manera, diagonalizando lo no lineal central a un kernel de caminata aleatoria, la descomposición espectral puede apresurar en gran medida su cómputo. El problema es que es computacionalmente indeseable ya que calcular la descomposición espectral de una matriz densa lleva un tiempo cúbico con respecto a su tamaño (Golub y Van Loan, 1996) y debido a que W_x es una matriz de $n^2 \times n^2$ esto resultaría en una complejidad de tiempo de $O(n^6)$ por cómputo de kernel.

Vishwanathan *et al.* (2010) muestran (Teorema 4) que se pueden obtener mucho mejor resultados -para grafos no etiquetados (aunque posiblemente con pesos en aristas)- haciendo uso de las propiedades del producto de Kronecker.

También se menciona que en la práctica, siempre se seleccionan series de potencias con límites conocidos que son triviales de evaluar (como $p = 1$), como se ejemplifica en el kernel geométrico (38) y en el kernel exponencial (39). Y es así como, a través del Teorema 4 de Vishwanathan *et al.* (2010), se muestra un método muy eficiente de computar matrices de kernels completas, aunque solo entre grafos no etiquetados.

III.2.3.5 Aproximación del producto de Kronecker más cercano

Los métodos rápidos para computar los kernels basados en caminatas aleatorias tratados por Vishwanathan *et al.* (2010) son computacionalmente muy costosos (en caso de ser aplicables) para grafos no etiquetados, especialmente si el número de etiquetas distintas d es grande o si se emplea un kernel de etiquetas general. Para estos casos, se puede encontrar el *producto de Kronecker más cercano* a W_x, esto es, computar las matrices S y T tal que $W_x \approx S \otimes T$, y después usar cualquiera de estos métodos sobre $S \otimes T$ como si se tratara

de la matriz de adyacencias del producto directo de grafos *no etiquetas*.

Existen algunos métodos para encontrar un producto de Kronecker que aproxime una matriz como W_x y que además sean eficientes al explotar la dispersión de W_x. Estos métodos minimizan la norma de Frobenius $||W_x - S \otimes T||_F$ al calcular el valor singular más grande de $\hat{W}_x$, que es una versión permutada de W_x. En este caso, el método utilizado por Vishwanathan *et al.* (2010) calcula, en cada iteración, el producto matriz-vector $\hat{W}_x \text{vec}(T')$ donde $T' \in \mathbb{R}^{n \times n}$ es la aproximación actual de T. Después, el resultado de ese producto matriz-vector, se reforma en una matriz de $n \times n$ que es la T' de la siguiente iteración. Calcular $\hat{W}_x \text{vec}(T')$ requiere $O(n^4)$.

Si W_x puede ser escrito como una suma de d productos de Kronecker (15) entonces también aplica para $\hat{W}_x$ (Van Loan, 2000) y su costo por iteración se reduce a $O(dn^2)$. Además, si dos grafos, con $O(n)$ aristas cada uno, son dispersos, W_x tendrá $O(n^2)$ elementos distintos a cero y cada iteración solo toma $O(n^2)$. El número de iteraciones k' requeridas es

$$k' = O\left(\frac{\ln n}{\ln |\xi_1| - \ln |\xi_2|}\right), \tag{40}$$

donde ξ_1 y ξ_2 es el eigenvalor de W_x de mayor magnitud y el segundo de mayor magnitud respectivamente.

El hecho de que la aproximación al producto de Kronecker más cercano se calcule por separado para cada uno de los elementos de la matriz del kernel de $m \times m$ causa que el método de la descomposición espectral tome $O(m^2 n^3)$ ya que no es posible precomputar el espectro de los m grafos. Además, la matriz del kernel resultante puede tener eigenvalores negativos lo que hace que el kernel deje de ser positivo semi-definido.

III.2.3.6 Deficiencias

Los kernels para grafos basados en caminatas aleatorias sufren de dos deficiencias que pueden afectar su desempeño así como su expresividad.

Primeramente, el fenómeno denominado *tottering* se refiere a cuando se presenta una

contribución desproporcionada al valor del kernel proveniente de caminatas cortas repetidas. Es decir, si consideramos el par de vértices adyacentes v y v', debido a este fenómeno las contribuciones al valor del kernel muchas veces se ven dominadas por caminatas de la forma $v \to v' \to v \cdots$.

Una solución a este problema sería diseñar algoritmos que evadan las caminatas aleatorias de este tipo. Desafortunadamente, un kernel de caminatas aleatorias auto-evasivas (del tipo que evita pasar por la misma arista dos veces) no puede ser computado en tiempo polinomial.

El segundo problema que tienen los kernels basados en caminatas aleatorias, denominado *halting*, está relacionado con el factor de descomposición $\mu(k)$ diseñado para evitar que la caminata aleatoria se extienda infinitamente. En la práctica, este factor siempre toma valores muy pequeños lo que tiene como consecuencia que los elementos correspondientes a caminatas largas sean despreciables.

III.3 Kernel basado en caminos más cortos

Como se vió anteriormente, el kernel de caminatas aleatorias sufre del problema de *halting* y *tottering*. Estos problemas son inherentes a este kernel debido a la naturaleza de las caminatas, es decir, al hecho de que las caminatas permiten ciclos y además se pueden extender indefinidamente y es debido a esto que se implementan medidas para lidiar con el problema pero sacrificando en expresividad y tiempo de cómputo.

Los *caminos* por otro lado, no admiten ser formadas por aristas repetidas y por consiguiente no es posible formar ciclos eliminando así el problema de *tottering*. Además, por la misma razón, no es posible tener un camino infinito, lo que ocasionaría *halting*, por lo cual no es necesario tener un factor de "descomposición" para controlar la logitud del camino.

III.3.1 Kernel de todos los caminos

La idea de usar *caminos* en vez de *caminatas* para evitar estos dos problemas es tratada por Borgwardt y Kriegel (2005) y Vishwanathan *et al.* (2010) en donde primeramente definen un kernel para grafos que utiliza el conjunto de todos los caminos que existen entre cada par de vértices de los grafos a tratar llamado *kernel de todos los caminos*.

Definición 3. Kernel de todos los caminos. Dados dos grafos G_1 y G_2. Sea $P(G_i)$ el conjunto de todos los caminos en el grafo G_i donde $i \in \{1, 2\}$. Sea $k_{caminos}$ un kernel positivo definido sobre dos caminos, se define como el producto de los kernels sobre aristas y vértices a lo largo del camino. Se define entonces el $k_{todos\ caminos}$ como

$$k_{todos\ caminos}(G_1, G_2) = \sum_{p_1 \in P(G_1)} \sum_{p_2 \in P(G_2)} k_{caminos}(p_1, p_2),$$

que es la suma de todos los kernels de pares de caminos de G_1 y G_2.

Además de esta definición, *Borgwardt y Kriegel (2005)* demuestran que el *kernel de todos los caminos* es positivo definido a través del siguiente Lemma:

Lemma 1. El kernel de todos los caminos es positivo definido.

Demostración: Se define una relación $R(p, G\{p\}, G)$, donde $G\{p\}$ y G son grafos y p es un camino. $R(p, G\{p\}, G) = 1$ si y solo si $G\{p\}$ es un grafo que se creó al remover de G todas las aristas y vértices en p.

$R^{-1}(G)$ es entonces un conjunto de todas las descomposiciones del grafo G via R a p y $G\{p\}$. R es finito, ya que existe un número finito de caminos en un grafo debido a que su longitud máxima está acotada por el número de aristas.

Se define el kernel sobre caminos k_{camino} como el producto de kernels en vértices y aristas de estos caminos, lo que es un *kernel de producto tensor* positivo definido (Schölkopf y Smola, 2002). También se define un kernel trivial $k_{uno} = 1$ para todo par de grafos.

Se define entonces un *kernel de todos los caminos* como un kernel de convolución-R

positivo definido (Haussler, 1999) :

$$k_{todos\ caminos}(G_1, G_2) = \sum_{R^{-1}(G_1)} \sum_{R^{-1}(G_2)} k_{caminos}(p_1, p_2) * k_{uno}(G\{p\}_1, G\{p\}_2)$$

$$= \sum_{p_1 \in (G_1)} \sum_{p_2 \in (G_2)} k_{camino}(p_1, p_2)$$

con $P(G_i)$ como el conjunto de todos los caminos en G_i, $i \in \{1, 2\}$.

Sin embargo, el cálculo de este kernel es *NP-difícil* como se demuestra en el siguiente lemma:

Lemma 2. Calcular el kernel de todos los caminos es NP-difícil.

Demostración: Se muestra este resultado demostrando que encontrar todos los caminos de un grafo es NP-difícil. Si no fuera NP-difícil determinar el conjunto de todos los caminos en un grafo, uno pudiera determinar si es que dicho grafo tiene un camino hamiltoniano[1] o no al revisar si es que existe un camino de longitud *n-1*.

Sin embargo, es conocido que este último problema es *NP-completo* (Jungnickel, 2008). Consecuentemente, determinar el conjunto de todos los caminos es *NP-difícil* y por lo tanto el cálculo del *kernel de todos los caminos* es NP-difícil.

Gärtner *et al.* (2003) muestran que los kernels basados en *subgrafos* es NP-difícil. En este caso se restringen a *caminos*, que son un subconjunto de estos subgrafos y sin embargo, su cálculo sigue siendo NP-difícil.

III.3.2 Kernel de caminos más cortos

Al no poder utilizar el *kernel de todos los caminos* debido a la complejidad en su cálculo, se puede considerar reducir el tamaño del conjunto de caminos al utilizar un subconjunto de todos ellos.

[1] **Camino hamiltoniano:** sucesión de aristas adyacentes que visita todos los vértices de un grafo solo una vez.

Se podría considerar el subconjunto que contiene *los caminos más largos* de un grafo pero, al igual que el kernel de todos los caminos, calcular los *caminos más largos* de un grafo es NP-difícil ya que permitiría decidir si existe o no un *camino hamiltoniano* en dicho grafo.

Por otro lado, calcular los *caminos más cortos* es un problema que se puede resolver en tiempo polinomial. Para esto. existen varios algoritmos ampliamente conocidos como *Dijkstra* (Dijkstra, 1959) que calcula los caminos más cortos desde un solo vértice y *Floyd-Warshall* (Floyd, 1962) para calcular todos los caminos más cortos en un grafo los cuales y permiten determinar las distancias más cortas en $O(m + n * logn)$ (Fredman y Tarjan, 1987) y $O(n^3)$ (Cormen *et al.*, 2001) respectivamente en donde m es el número de aristas y n es el número de vértices.

Una vez que se decidió utilizar los caminos más cortos en la aplicación de un kernel para grafos, surge un problema: ¿Cómo se pueden utilizar los caminos más cortos como medida de similitud entra grafos si los caminos más cortos entre dos vértices no son únicos? Es decir, puede existir más de un camino más corto entro un par de vértices de un grafo como se muestra en la Figura 10.

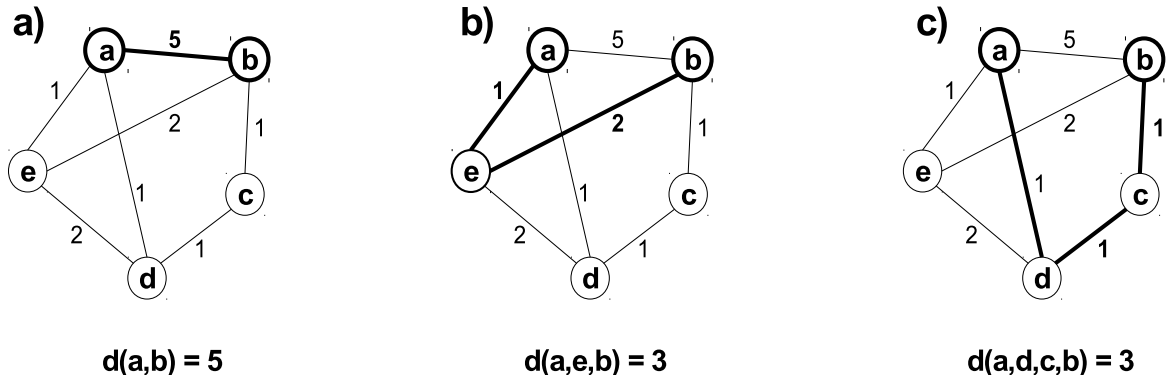


Figura 10. Tres caminos distintos del vértice ***a*** al vértice ***b***. Los caminos en *b)* y *c)* son caminos más cortos. ***a)*** La arista **(*a*, *b*)** con un costo de 5. ***b)*** Camino de ***a*** a ***b*** pasando por ***e*** con un costo total de 3. ***c)*** Camino de ***a*** a ***b*** pasando por ***d*** y ***c*** con un costo total también de 3.

Borgwardt y Kriegel (2005) proponen utilizar la medida de distancia de estos caminos ya que, mientras es cierto que puede existir más de un camino más corto entre un par de

vértices, la distancia mínima es la misma lo que se demuestra al plantear que, en el caso de que uno de estos caminos fuera más corto, el otro no pudiera ser considerado como camino más corto.

En este caso, se emplean 3 características únicas de los caminos más cortos para calcular el kernel: el vértice inicial, el vértice final y la distancia entre ambos.

La matriz que contiene las distancias de los caminos más cortos entre cada par de vértices de un grafo es comunmente llamada *matriz de distancias de caminos más cortos* SP y se define como

$$SP_{ij} = \begin{cases} d(v_i, v_j) & si\ v_i\ y\ v_j\ están\ conectados, \\ \infty & otro\ caso \end{cases}$$

en donde $d(v_i, v_j)$ es la distancia del camino más corto entre v_i y v_j.

Ya que para la definición del kernel para grafos basado en caminos más cortos es necesaria la comparación por pares de cada uno de los elementos finitos de SP, que son las distancias más cortas entre cada par de vértices de G, se puede crear un grafo que contenga solo esa información, es decir, que tenga el mismo conjunto de vértices que el grafo original pero que exista una arista entre todos los vértices que estén conectados a través de un camino. Este grafo S puede ser descrito por su matriz de adyacencias que se formaría de la siguiente manera:

$$A(S)_{ij} = \begin{cases} 1 & si\ SP(v_i, v_j) < \infty, \\ 0 & otro\ caso \end{cases}$$

en donde $SP(v_i, v_j)$ es la etiqueta de la arista $(v_i, v_j) \in E$ del grafo de caminos más cortos S.

III.3.2.1 Transformación Floyd

El primer paso para obtener el *kernel de caminos más cortos* es transformar el grafo original en un *grafo de caminos más cortos*. Un grafo de caminos más cortos S contiene los mismos vértices que el grafo original G pero, a diferencia del grafo de entrada, existe una arista

entre cada par de vértices conectados por una caminata en G. Cada arista en S entre los vértices v_i y v_j está etiquetada con la distancia más corta entre estos dos vértices.

Existen varios algoritmos que se pueden aplicar para resolver el problema de los caminos más cortos entre todos los pares y así determinar las aristas de distancias más cortas en S. Borgwardt y Kriegel (2005) proponen utilizar el *algoritmo de Floyd-Warshall* (Algoritmo 1). Este algoritmo tiene una tiempo de ejecución de $O(n^3)$ (Cormen *et al.*, 2001) y además es fácil de implementar. Su aplicación no se restringe a grafos con aristas negativas siempre y cuando no contengan ciclos negativos. Al proceso de transformación de un grafo G a uno S por medio del algoritmo Floyd se le denomina *transformación Floyd*.

Algoritmo 1 Pseudocódigo del algoritmo Floyd-Warshall para encontrar la matriz de caminos más cortos entre todos los pares (extraido de Borgwardt (2007), 59).

Entrada: Matriz de adyacencias A de grafo G de n vértices y los pesos de aristas w.
Salida: Matriz de distancias de caminos más cortos SP.

```
1  for i = 1 to n do
2      for j = 1 to n do
3          if A[i,j] == 1 and i ≠ j then
4              SP[i,j] = w[i,j]
5          else
6              if i == j then
7                  SP[i,j] = 0
8              else
9                  SP[i,j] = ∞
10             end if
11         end if
12     end for
13 end for
14 for k = 1 to n do
15     for i = 1 to n do
16         for j = 1 to n do
17             if SP[i,k] + SP[k,j] < SP[i,j] then
18                 SP[i,j] = SP[i,k] + SP[k,j]
19             end if
20         end for
21     end for
22 end for
```

III.3.3 Definición del kernel

Después de la *transformación Floyd* de los grafos de entrada, se puede definir el *kernel de caminos más cortos*:

Definición 4. Kernel de caminos más cortos

Sea S y S' dos grafos de caminos más cortos. Se define el kernel de caminos más cortos sobre $S = (V, E)$ y $S' = (V', E')$ como

$$k_{caminos\ más\ cortos}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k^1_{caminata}(e, e'), \tag{41}$$

en donde $k^1_{caminata}$ es un kernel de caminatas aleatorias de longitud 1.

El kernel de caminos más cortos requiere de la transformación de *Floyd* que puede ser calculada en $0(n^3)$ cuando se utiliza el algoritmo de *Floyd-Warshal*. Si el grafo original es conectado, el número de aristas del grafo transformado es n^2. Ya que es necesario la comparación por pares de todas las aristas de ambos grafos transformados para obtener el valor del kernel, se deben de considerar $n^2 * n^2$ pares de aristas lo que resulta en un tiempo de ejecución total de $O(n^4)$.

Borgwardt *et al.* (2005) demuestran la validez del *kernel de caminos más cortos* con el siguiente lemma:

Lemma 3. *El kernel de caminos más cortos es positivo definido*

Demostración: El kernel de caminos más cortos es positivo definido

Borgwardt *et al.* (2005) modifican el kernel de caminatas aleatorias y definen un kernel que llaman simplemente "Kernel de caminatas aleatorias modificado". Este kernel se puede definir de la siguiente manera:

Sean $G = (V, E)$ y $G' = (V', E')$ dos grafos. Considere dos caminatas, $caminata_1 = (v_1, v_2, \cdots, v_{n-1}, v_n)$ en G y $caminata_2 = (v'_1, v'_2, \cdots, v'_{n-1}, v'_n)$ en G' donde $v \in V$, $v' \in V'$ para $i \in \{1, \cdots, n\}$ y $(v_i, v_{i+1}) \in E$, $(v'_i, v'_{i+1}) \in E'$ para $i \in \{1, \cdots, n-1\}$. El kernel de

caminatas se define como:

$$k_{caminata}(caminata_1, caminata_2) = \prod_{i=1}^{n-1} k_{paso}((v_i, v_{i+1}), (v'_i, v'_{i+1})). \tag{42}$$

Este kernel se basa en 3 características principales en la caminata: el vértice de origen, el vértice destino y la arista entre ellos. k_{paso} se define de la siguiente manera:

Para $i \in \{1, \cdots, n-1\}$,

$$k_{paso}((v_i, v_{i+1}), (v'_i, v'_{i+1})) = k_{vértice}(v_i, v'_i) * k_{vértice}(v_{i+1}, v'_{i+1}) * k_{arista}((v_i, v_{i+1}), (v'_i, v'_{i+1})). \tag{43}$$

La definición de las partes de k_{paso} ($k_{vértice}$ y k_{arista}) depende de varios factores a tomar en cuenta. Estos factores pueden ser el hecho de usar o no etiquetas en vértices y/o aristas, si se busca tener una función binaria de similitud o se desea algún factor que represente que tanto se "parecen" las caminatas, etc.

Dadas las definiciones 41 y 42 y debido a que el kernel de *caminos más cortos* es un kernel de caminatas aleatorias de longitud 1, tenemos que:

$$k_{caminos\ más\ cortos}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k_{paso}(e, e'), \tag{44}$$

donde S y S' dos grafos de caminos más cortos, $S = (V, E)$ y $S' = (V', E')$ como en 41.

Construcción del kernel de caminos más cortos

Cómo se describe en el capítulo II, es posible crear kernels a partir de otros más simples. Borgwardt *et al.* (2005) utilizan los kernels *Dirac*, *RBF* y *Puente Browniano* (4, 5 y 6) en su implementación de los kernels aplicados en la predicción de funciones de proteínas mientras que Kashima *et al.* (2003) solo mencionan el kernel *RBF* y *Dirac* (5 y 4) y proponen el uso de uno solo de acuerdo a las necesidades tanto para la comparación de aristas y de vértices.

III.3.4 Enriquecimiento de etiquetas

Un problema que existe con el cálculo del kernel basado en caminos más cortos es el hecho de que, a través de la transformación de *Floyd* que sufren, los grafos de entrada se conviertan en grafos densos, aunque inicialmente sean dispersos (considerando que sean grafos conectados, el grafo de salida sería un grafo completamente conectado). Debido a esto, el proceso de comparación por pares de las aristas se vuelve muy costoso.

Además de esto, se puede considerar que el valor de uno de los caminos más cortos entre vértices es poca información para poder establecer una medida de similitud entre grafos ya que, como se muestra en la figura 10, aunque la longitud del camino más corto es única, puede existir más de una forma de llegar de un vértice al otro con el costo mínimo.

Mahé *et al.* (2004) propusieron una técnica llamada *enriquecimiento de etiquetas* pensando en solucionar estos problemas. Esta técnica consiste en obtener más información de un proceso para así contribuir a la distinción de estos objetos con más facilidad, en este caso, en el proceso de la transformación de *Floyd* (algoritmo 2).

La técnica de enriquecimiento de etiquetas, además sirve para mejorar el tiempo de procesamiento ya que, basado en la información extra obtenida tanto de vértices como de aristas, se puede evitar realizar ciertos procesos lo cual resulta en un ahorro en tiempo.

Con esto en mente, Borgwardt y Kriegel (2005) propusieron el uso del *enriquecimiento de etiquetas* en su *kernel de caminos más cortos de longitudes iguales* el cual toma en cuenta el número de pasos recorridos de i a j y de i' a j' y solo en el caso de que sean iguales regresa un 1, es decir, solo en caso de que los caminos tengan el mismo número de pasos se calcula el valor del kernel tomando en cuenta la distancia total.

De esta manera, el kernel de *caminos más cortos de longitudes iguales* al que simplemente se le llamará kernel de *caminos más cortos* (k_{SP}) se construye de la siguiente manera:

$$k_{SP}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k_{long}(e, e') k_{paso}(e, e'), \tag{45}$$

donde S y S' son dos grafos de caminos más cortos, $S = (V, E)$ y $S' = (V', E')$.

Algoritmo 2 Pseudocódigo del algoritmo Floyd-Warshall para encontrar la matriz de caminos más cortos entre todos los pares modificado para extraer más información.

Entrada: Matriz de adyacencias A de grafo G de n vértices y los pesos de aristas w.

Salida: Matriz de distancias de caminos más cortas SP, matriz de número de pasos St y matriz de caminos más cortos P.

```
1  for i = 1 to n do
2      for j = 1 to n do
3          if A[i,j] == 1 and i ≠ j then
4              SP[i,j] = w[i,j]
5          else
6              if i == j then
7                  SP[i,j] = 0
8              else
9                  SP[i,j] = ∞
10             end if
11         end if
12     end for
13 end for
14 for k = 1 to n do
15     for i = 1 to n do
16         for j = 1 to n do
17             if SP[i,k] + SP[k,j] < SP[i,j] then
18                 SP[i,j] = SP[i,k] + SP[k,j]
19                 St[i,j] = St[i,j] + 1
20                 P[i,j] = CONCATENAR(P[i,j], w[i,j])
21             end if
22         end for
23     end for
24 end for
```

De las partes que componen a k_{SP}, k_{long} se definen de la siguiente manera:

$$k_{long}(e, e') = k_{Dirac}(longitud(e), longitud(e'))), \tag{46}$$

en donde $longitud(e)$ es la longitud (en número de pasos) entre los dos extremos de la arista e. Esta información se extrae con el proceso de enriquecimiento de etiquetas y a nivel código en su implementación, funciona como un criterio de descarte en caso de no tener longitudes iguales. Es decir, en caso de que las dos longitudes de e y e' no sean iguales, se considera ese elemento como un 0 y se procede al siguiente elemento.

Por último, de acuerdo a 43, k_{paso} está compuesto por $k_{vértice}(v, v')$ y $k_{arista}(e, e')$ los cuales se definen de la siguiente manera:

$$k_{vértice}(v, v') = k_{Dirac}(v, v'). \tag{47}$$

En la sección III.5 se pronen 3 definiciones distintas de k_{arista} como variantes del kernel de *caminos más cortos*.

III.4 Kernel basado en subárboles de Ramon-Gärtner

Definición 5. El kernel de sub-árboles original fue definido por Ramon y Gärtner (2003). Compara todos los pares de nodos de los grafos $G = (V, E, \mathcal{L})$ y $G' = (V', E', \mathcal{L}')$ comparando iterativamente sus vecindarios:

$$k^{(h)}_{Ramon}(G, G') = \sum_{v \in V} \sum_{v' \in V'} k_h(v, v'), \tag{48}$$

donde

$$k_h(v, v') = \begin{cases} \delta(\mathcal{L}(v), \mathcal{L}'(v')), & si\ h = 1 \\ \lambda_r \lambda_s \sum_{R \in \mathcal{M}(v,v')} \prod_{(w,w') \in R} k_{h-1}(w, w'), & si\ h > 1 \end{cases}, \tag{49}$$

y

$$\mathcal{M}(v, v') = \{R \subseteq \mathcal{N}(v) \times \mathcal{N}(v') | (\forall (u, u'), (w, w') \in R : u = w \Leftrightarrow u' = w') \\ \wedge (\forall (u, u') \in R : \mathcal{L}(u) = \mathcal{L}(u'))\}. \tag{50}$$

Es decir, k_{Ramon} compara iterativamente todos las coincidencias $\mathcal{M}(v, v')$ entre vecindarios de dos vértices v y v' de G y G' respectivamente.

Shervashidze *et al.* (2011) demuestran el vínculo del kernel de sub-árboles de *Ramon-Gärtner* (48) con el kernel de sub-árboles de *Weisfeiler-Lehman* (52) definiendo el segundo de manera recursiva para demostrar su equivalencia.

III.4.1 La prueba de isomorfismo de Weisfeiler-Lehman

El algoritmo de Shervashidze *et al.* (2011) para calcular el kernel de sub-árboles rápido hace uso de la prueba de isomorfismo de *Weisfeiler-Lehman* (Weisfeiler y Lehman, 1968), específicamente su variante de una dimensión, conocida como "refinamiento ingenuo de vértices".

Suponiendo que se tienen dos grafos G y G' y se quiere saber si son isomórficos o no, la prueba de *Weisfeiler-Lehman* comprende de iterar los pasos del algoritmo 3 un número h de veces.

La idea principal del algoritmo es aumentar las etiquetas de los vértices con el conjunto ordenado de etiquetas de los vértices vecinos y comprimir esta etiqueta nueva "aumentada" en forma de nuevas etiquetas más cortas. Estos pasos se repiten hasta que los conjuntos de etiquetas de vértices de G y G' difieran o hasta que el número de iteraciones llegue a n. La figura 11 muestra una ilustración de los pasos de una iteración del algoritmo.

El paso 3 del algoritmo facilita la definición e implementación de la función f para la compresión de etiquetas en el paso 4: se define una variable contador para f que lleva la cuenta de cuantas cadenas distintas ha comprimido la función f hasta el momento. f asigna el número actual del contador a una cadena si una cadena idéntica ha sido comprimida

Algoritmo 3 Una iteración de la prueba de isomorfismo de grafos Weisfeiler-Lehman de una dimensión (extraido de Shervashidze y Borgwardt (2009), 3)

1: Determinación de etiqueta-multiconjunto

- Para $h = 1$, hacer $M_h(v) := l_0(v) = \mathcal{L}(v)$ para grafos etiquetados, y $M_h(v) := l_0(v) = |\mathcal{N}(v)|$ para grafos no etiquetados.
- Para $h > 1$, asignar una etiqueta-multiconjunto $M_h(v)$ a cada vértice v en G y G' que consiste del multiconjunto $\{l_{h-1}(u)|u \in \mathcal{N}(v)\}$.

2: Ordenamiento de cada multiconjunto

- Ordenar los elementos de $M_h(v)$ en orden ascendente y concatenarlos a una cadena $s_h(v)$.
- Sumar $l_{h-1}(v)$ como prefijo a $s_h(v)$.

3: Ordenamiento del conjunto de multiconjuntos

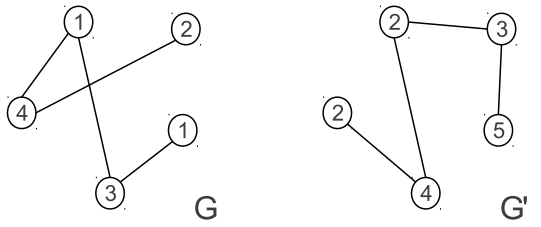
- Ordenar todas las cadenas $s_h(v)$ para todo v de G y G' en orden ascendente.

4: Compresión de etiquetas

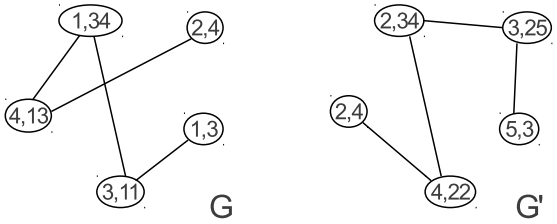
- Mapear cada cadena $s_h(v)$ a una etiqueta comprimida nueva, usando una función $f : \sum^* \to \sum$ tal que $f(s_h(v)) = f(s_h(w))$ si y solo si $s_h(v) = s_h(w)$.

5: Re-etiquetado

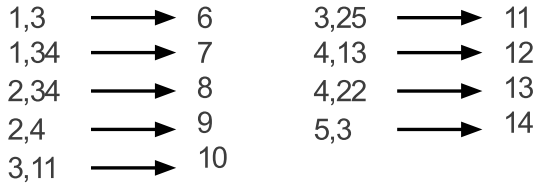
- Hacer $l_h(v) := f(s_h(v))$ para todos los vértices en G y G'.



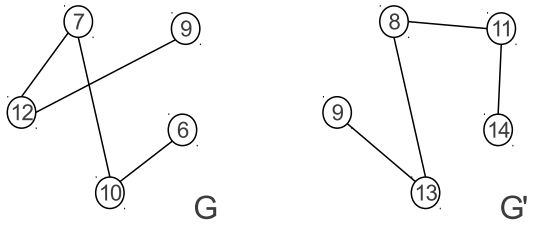
(a) Grafos de entrada G y G'.



(b) Resultado de los pasos 1 y 2: determinación de etiquetas multiconjunto y ordenamiento.



(c) Resultado del paso 3: compresión de etiquetas.



(d) Resultado del paso 4: re-etiquetado.

Figura 11. Ilustración del cálculo del kernel de sub-árboles WL con h = 1 (primera iteración) para dos grafos.

$$\phi^{(1)}_{subárbolesWL}(G) = (\ 2,1,1,1,0\ ,\ 1,1,0,1,1,0,1,0,0)$$
$$\phi^{(1)}_{subárbolesWL}(G') = (\ \underbrace{0,2,1,1,1}_{\substack{\text{Conteo de etiquetas}\\ \text{de vértices originales}}}\ ,\ \underbrace{0,0,1,1,0,1,0,1,1}_{\substack{\text{Conteo de etiquetas}\\ \text{de vértices comprimidas}}})$$

$$k^{(1)}_{subárbolesWL}(G, G') = \langle \phi^{(1)}_{subárbolesWL}(G), \phi^{(1)}_{subárbolesWL}(G') \rangle = 5$$

Figura 12. Representación de los vectores de características de G y G' y cálculo del producto interno del ejemplo de la Figura 11.

anteriormente, en caso de no ser así, se incrimenta el contador en 1 y asigna el nuevo valor a la etiqueta nueva. El hecho de que estén ordenadas las etiquetas multiconjunto (paso 3) garantiza que todas las cadenas idénticas son mapeadas al mismo número ya que ocurren en bloques consecutivos.

El algoritmo *Weisfeiler-Lehman* termina después del paso 5 de la iteración h si $\{l_h(v)|v \in V\} \neq \{l_h(v')|v' \in V'\}$, esto es, si los conjuntos de etiquetas recién creadas no son idénticas en G y G'. En ese caso, los grafos no son *isomórficos*.

Por otro lado, si los conjuntos siguen siendo idénticos después de la iteración n, existen dos posibilidades: los grafos G y G' son isomórficos o el algoritmo no pudo determinar si lo son en el número de iteraciones.

Shervashidze *et al.* (2011) demuestran como, a través del uso de los algoritmos "Counting sort" y "Radix sort" (Cormen *et al.*, 2001) para el ordenamiento de cada multiconjunto (dado el rango limitado de elementos del multiconjunto) y las cadenas multiconjunto respectivamente, el tiempo de ejecución del algoritmo *Weisfeiler-Lehman* de una dimensión para h iteraciones es $O(hm)$.

Cabe destacar que la relación entre la prueba *Weisfeiler-Lehman* con los patrones de sub-árboles se puede ver en la figura 13 en donde las etiquetas comprimidas $l_i(v)$ corresponden a patrones de sub-árboles de altura h enraizado en v.

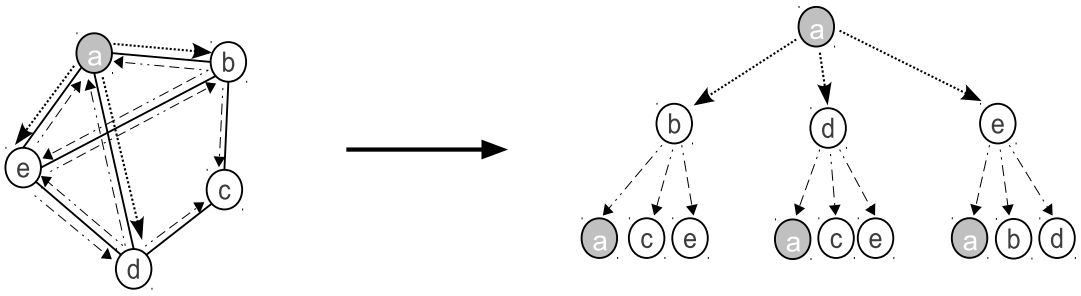


Figura 13. Un patrón de sub-árboles de altura 2 enraizado en el vértice **a** ($l_0(v) = a$ en este caso). Note como se repiten los vértices al "desdoblarse" el patrón de sub-árboles.

III.4.2 El kernel Weisfeiler-Lehman general

III.4.2.1 El marco de trabajo del kernel Weisfeiler-Lehman

En cada iteración h del algoritmo de *Weisfeiler-Lehman* (algoritmo 3), obtenemos un nuevo etiquetado $l_i(v)$ para todos los vértices v. Recordando que si los vértices en G y G' tienen etiquetas multiconjunto idénticas, solo en ese caso, tendrán nuevas etiquetas idénticas. Por lo tanto, se puede ver a una iteración del reetiquetado *Weisfeiler-Lehman* como una función $w((V, E, l_i)) = (V, E, l_{i+1})$ que transforma todos los grafos de la misma manera.

Definición 6. Se define al kernel para grafos Weisfeiler-Lehman a una altura i del grafo $G = (V, E, \ell) = (V, E, l_0)$ como el grafo $G_i = (V, W, l_i)$. Se le llama a la secuencia de grafos de *Weisfeiler-Lehman*

$$\{G_0, G_1, \cdots, G_h\} = \{(V, E, l_0), (V, E, l_1), \cdots, (V, E, l_h)\},$$

donde $G_0 = G$ y $l_0 = \ell$, la secuencia Weisfeiler-Lehman hasta una altura h de G.

G_0 es el grafo original $G_1 = w(G_0)$ es el grafo resultante del primer re-etiquetado y así sucesivamente.

Definición 7. Sea k cualquier kernel para grafos, que se le llamará *kernel base*. Entonces el kernel *Weisfeiler-Lehman* con h iteraciones con el kernel base se define como

$$k_{WL}^{(h)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \cdots + k(G_h, G'_h), \tag{51}$$

donde h es el número de iteraciones de Weisfeiler-Lehman y $\{G_0, \cdots, G_h\}$ y $\{G'_0, \cdots, G'_h\}$ son las secuencias de Weisfeiler-Lehman de G y G' respectivamente.

Lemma 4. Sea el kernel base k cualquier kernel para grafos positivo semidefinido. Entonces, el kernel Weisfeiler-Lehman $k_{WL}^{(h)}$ correspondiente es positivo semidefinido.

Demostración Sea ϕ el mapa de características correspondiente al kernel k:

$$k(G_i, G'_i) = \langle \phi(G_i), \phi(G'_i) \rangle.$$

Tenemos

$$k(G_i, G'_i) = k(w^i(G), w^i(G')) = \langle \phi(w^i(G)), \phi(w^i(G')) \rangle.$$

Se define el mapa de características $\psi(G)$ como $\phi(w^i(G))$. Entonces tenemos

$$k(G_i, G'_i) = \langle \psi(G), \psi(G') \rangle,$$

por lo tanto k es un kernel sobre G y G' y $k_{WL}^{(h)}$ es positivo semidefinido como la suma de kernels positivos semidefinidos.

III.4.3 El kernel de sub-árboles de Weisfeiler-Lehman

Basándose en el algoritmo 3 de *Weisfeiler-Lehman*, Shervashidze y Borgwardt (2009) definen la siguiente función kernel que es una instancia natural de (51).

Definición 8. Sea G y G' dos grafos. Se define $\sum_i \subseteq \sum$ como un conjunto de letras que ocurren como etiquetas de los vértices al menos una vez en G o G' al final de la iteración número h del algoritmo *Weisfeiler-Lehman*. Sea $\sum_0$ el conjunto de etiquetas originales de vértices de G y G'. Asumiendo todos los pares de $\sum_i$ son disjuntos. Sin pérdida de generalidad, se asume que cada $\sum_i = \{\sigma_{i1}, \cdots, \sigma_{i|\sum_i|}\}$ está ordenada.

Se define el mapa $c_i : \{G, G'\} \times \sum_i \to \mathbb{N}$ tal que $c_i(G, \sigma_{ij})$ es el número de ocurrencias de la letra σ_{ij} en el grafo G.

Se define al kernel de sub-árboles *Weisfeiler-Lehman* sobre dos grafos G y G' con h iteraciones como:

$$k^{(h)}_{subárbolesWL}(G, G') = \langle \phi^{(h)}_{subárbolesWL}(G), \phi^{(h)}_{subárbolesWL}(G') \rangle, \tag{52}$$

donde

$$\phi^{(h)}_{subárbolesWL}(G) = (c_0(G, \sigma_{01}), \cdots, c_0(G, \sigma_{0|\sum_0|}), \cdots, c_h(G, \sigma_{h1}), \cdots, c_h(G, \sigma_{h|\sum_h|}))$$

y

$$\phi^{(h)}_{subárbolesWL}(G') = (c_0(G', \sigma_{01}), \cdots, c_0(G', \sigma_{0|\sum_0|}), \cdots, c_h(G', \sigma_{h1}), \cdots, c_h(G', \sigma_{h|\sum_h|})).$$

Esto es, el kernel de sub-árboles de *Weisfeiler-Lehman* cuenta etiquetas *originales* y *comprimidas* comunes en dos grafos (como se ilustra en las Figuras 11 y 12).

Este algoritmo, como lo demuestran Shervashidze *et al.* (2011), puede ser calculado en $O(hm)$.

Shervashidze *et al.* (2011) demuestran con el siguiente lemma que el kernel es positivo definido al demostrar que el kernel de sub-árboles de *Weisfeiler-Lehman* (52) es de hecho un caso especial del kernel general *Weisfeiler-Lehman* (51).

Lemma 5. Sea el kernel base k una función que cuenta pares de etiquetas de vértices iguales en dos grafos:

$$k(G, G') = \sum_{v \in V} \sum_{v' \in V'} \delta(\ell(v), \ell(v')),$$

en donde δ es el kernel Dirac (4), esto es, es 1 cuando sus argumentos son iguales y 0 en caso contrario.

Entonces $k^{(h)}_{WL}(G, G') = k^{(h)}_{subárbolesWL}(G, G')$ para todos G, G'.

Demostración Se puede notar que para cada $i \in \{0, 1, \cdots, h\}$ tenemos

$$\sum_{v \in V} \sum_{v' \in V'} \delta(\ell_i(v), \ell'_i(v')) = \sum_{j=1}^{|\sum_i|} c_i(G, \sigma_{ij}) c_i(G', \sigma_{ij}).$$

Sumando estas sumatorias para todo $i \in \{0, 1, \cdots, h\}$ resulta que

$$k^{(h)}_{WL}(G, G') = k^{(h)}_{subárbolesWL}(G, G').$$

III.4.4 Kernel de sub-árboles Weisfeiler-Lehman sobre *N* grafos

Como calcular el kernel de sub-árboles *Weisfeiler-Lehman* (ecuación 52) sobre un par de grafos G y G' toma un tiempo de $O(hm)$, para un conjunto de N grafos, la matriz *kernel* de $N \times N$ elementos se calcula en $O(N^2hm)$.

En vez del cálculo "ingenuo" del kernel, es decir, ejecutar el kernel de sub-árboles *Weisfeiler-Lehman* para cada uno de los N^2 pares del conjunto de datos de N elementos, Shervashidze *et al.* (2011) proponen procesar los N grafos simultaneamente y aplicar los pasos dados en el algoritmo 4 para cada grafo G en cada iteración h.

Algoritmo 4 Una iteración del kernel Weisfeiler-Lehman para N grafos (extraido de Shervashidze y Borgwardt (2009),5)

1: Determinación de la etiqueta multiconjunto

- Asignar una etiqueta-multiconjunto $M_h(v)$ a cada vértice v en G y G' que consiste del multiconjunto $\{l_{h-1}(u)|u \in \mathcal{N}(v)\}$.

2: Ordenamiento de cada multiconjunto

- Ordenar los elementos de $M_h(v)$ en orden ascendente y concatenarlos a una cadena $s_h(v)$.
- Sumar $l_{h-1}(v)$ como prefijo a $s_h(v)$.

3: Compresión de etiquetas

- Mapear cada cadena $s_h(v)$ a una etiqueta comprimida nueva, usando una función $f : \sum^* \to \sum$ tal que $f(s_h(v)) = f(s_h(w))$ si y solo si $s_h(v) = s_h(w)$.

4: Re-etiquetado

- Hacer $l_h(v) := f(s_h(v))$ para todos los vértices en G y G'.

En este caso, se asume que $\sum$ es lo suficientemente grande para permitir que f sea inyectiva. Para el caso de N grafos y h iteraciones, con un $\sum$ de tamaño $Nn(h+1)$ es suficiente.

Una forma de implementar f es ordenar todas las cadenas del vecindario usando "Radix sort", como se hace en el paso 5 del algoritmo 3.

Como alternativa a la implementación de f puede ser mediente el uso de una función hash *perfecta*[2] como lo sugieren Shervashidze y Borgwardt (2009). Esta función hash f puede ser implementada eficientemente: guarda una variable contador x que cuenta el número de cadenas distintas que f ha mapeado a una etiqueta comprimida hasta ese punto. Si se aplica f a una cadena distinta a todas las anteriores, entonces se incrementa x en uno y se mapea dicha cadena a x.

Lemma 6. Para N grafos, el kernel de sub-árboles *Weisfeiler-Lehman* con h iteraciones sobre todos los pares de estos grafos puede ser calculado en $O(Nhm + N^2hn)$.

Demostración La aplicación "ingenua" del kernel de sub-árboles *Weisfeiler-Lehman* (52) para calcular la matriz de kernel de $N \times N$ requeriría un tiempo de ejecución de $O(N^2hm)$. Esto se puede mejorar calculando $\phi^{(h)}_{subárbolesWL}$ explícitamente para cada grafo y solo entonces calcular el producto interno (como se muestra en la figura 12).

Paso 1, la determinación de las etiquetas multiconjunto siguen tomando $O(Nm)$. Paso 2, el ordenamiento de los elementos de cada multiconjunto puede ser realizado mediante *counting sort* de todas las cadenas, requiere un tiempo de $O(Nn + Nm)$.

El esfuerzo de calcular $\phi^{(h)}_{subárbolesWL}$ sobre todos los N grafos en h iteraciones es entonces $O(Nhm)$, asumiendo que $m > n$. Para obtener todos los valores por pares del kernel, se deben de multiplicar todos los vectores de características, lo que requiere de un tiempo de $O(N^2hn)$ ya que cada grafo G tiene máximo hn elementos distintos a cero en $\phi^{(h)}_{subárbolesWL}(G)$ (Shervashidze y Borgwardt, 2009).

III.5 Variantes propuestas

En el desarrollo y estudio de los kernels para grafos basados en *caminos más cortos* y *sub-árboles* para su comparación, se pudo indentificar una serie de áreas en las que se pueden

[2]función hash perfecta: función hash que mapea un conjunto de elementos a enteros sin provocar colisiones

implementar variaciones para buscar un mejor desempeño ya sea en tiempo de cálculo y/o en el porcentaje de clasificación.

Estas variaciones se dividen en dos grupos: las variaciones propuestas al kernel de *caminos más cortos* y las variaciones propuestas al kernel de *sub-árboles*.

III.5.1 Variantes del kernel de caminos más cortos

El proceso de transformar un grafo a uno de caminos más cortos, resulta efectivo pues el grafo resultante ofrece más características de las que tiene el grafo original, lo que regularmente se traduce a una mejor clasificación.

Sin embargo, esta transformación trae desventajas en el tiempo de ejecución por la naturaleza de los grafos de salida pues son grafos *densos*.

Por este motivo, se idean varias modificaciones o variantes de kernels que aprovechan los beneficios que trae la transformación de *Floyd-Warshall* y en algunos casos tratan de reducir el tiempo de ejecución.

III.5.1.1 Caminos más cortos de pasos iguales (*iSP*)

Además del kernel de *caminos más cortos* (k_{SP}) y siguiendo con la idea de ahorrar operaciones además de enriquecer el cálculo del kernel usando más información, se propone el uso del kernel de *caminos más cortos de distancias iguales* al que llamaremos kernel de *caminos más cortos invertido* (k_{iSP}) ya que, de cierta manera se invierten los criterios formación del kernel.

Como se indica en 45, k_{SP} se compone de dos kernels, k_{long} y k_{paso}. En este caso, k_{long} fue implementado como resultado del *enriquecimiento de etiquetas* pues utiliza como criterio la longitud (en número de pasos) de la arista en cada uno de los pasos para el cálculo del kernel k_{paso}. Es decir, si los números de pasos son distintos, el valor de k_{long} es igual a 0 y con eso cancela el valor de k_{paso}.

En caso de ser iguales, se procede a calcular $k_{paso_{iSP}}$ que a su vez está formado por

distintos kernels básicos (43). Este kernel procesa la información del *paso* de un vértice a otro, es decir, los *vértices* de inicio y fin, y las etiquetas de las *aristas* que los conectan.

El kernel que procesa los vértices tanto de inicio y fin ($k_{vértice}$) se definió en la sección III.3 en (47) y la definición de (k_{arista}) se hace más adelante al introducir 3 distintas opciones.

En este caso, para el kernel k_{iSP}, se utiliza la *distancia del camino más corto* obtenida del algoritmo de *Floyd-Warshall*, como criterio para considerar el cálculo de $k_{paso_{iSP}}$. Solo en caso de que el valor de ambas distancias sean iguales, se procede a calcular el kernel $k_{paso_{iSP}}$.

A continuación se definen estos kernels:

Sean $S = (V, E)$ y $S' = (V', E')$ dos grafos de caminos más cortos y $e = (v_i, v_{i+1})$ y $e' = (v'_i, v'_{i+1})$ para $i \in \{1, \cdots, n-1\}$ dos aristas donde $e \in E$, $e' \in E'$ y $v \in V$, $v' \in V'$,

$$k_{iSP}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k_{dist}(e, e') k_{paso_{iSP}}(e, e'), \tag{53}$$

en donde

$$k_{dist}(e, e') = k_{Dirac}(distancia(e), distancia(e')) \tag{54}$$

y

$$k_{paso_{iSP}}(e, e') = k_{vértice}(v_i, v'_i) * k_{vértice}(v_{i+1}, v'_{i+1}) * k_{arista}((longitud(e), longitud(e')) \tag{55}$$

donde $distancia(e)$ y $longitud(e)$ son la *distancia* y *longitud* (en número de pasos) de la arista e respectivamente, obtenidas con el algoritmo de *Floyd-Warshall* utilizando enriquecimiento de aristas (Algoritmo 2).

III.5.1.2 Conteo de caminos más cortos

La implementación del kernel de conteo de caminos más cortos (SPC) sigue la idea de Kramer y Readt (2001) con su kernel de "Descubrimiento de patrones" (*Pattern discovery*) o el propio kernel de sub-árboles *Weisfeiler-Lehman* de Shervashidze y Borgwardt (2009) en donde construye vectores de características contando cierto patrón o cadena en los grafos a clasificar.

En este caso, es el proceso de *enriquecimiento de etiquetas* usado en la aplicación del kernel de *caminos más cortos*, en donde se obtiene más información del grafo al momento de aplicar la transformación de *Floyd-Warshall*. En este proceso, además de sacar solamente la distancia del camino más corto entre cada par de vértices en un grafo (como se hace normalmente), se obtiene una cadena que contiene la etiqueta de cada vértice por el que se pasa en dicho camino más corto entre cada par de vértice (matriz P que obtiene de salida el algoritmo 2).

Con esta información, se construye los vectores de características contando las apariciones de cada una de estas cadenas en los grafos de entrada, de ahí el nombre de *conteo de caminos más cortos* (*SPC* por sus siglas en inglés).

III.5.2 Variantes del kernel de sub-árboles

Sin duda, el método kernel de sub-árboles propuesto por Shervashidze *et al.* (2011) es muy rápido a comparación de otros métodos existentes. Esto se debe a la forma en la cual construye los vectores de características de cada uno de los grafos, a través del algoritmo 4, para posteriormente realizar el cálculo (como se ilustra en la Figura 12).

En esta sección se proponen una serie de variaciones a este kernel tratando de conservar esa velocidad y principalmente buscando mejorar el desempeño en porcentaje de clasificación. Esto se hace combinando distintos kernels más básicos, modificando el propio algoritmo de *Weisfeiler-Lehman* para tomar en cuenta las aristas o transformando los grafos de entrada antes de ser procesados.

A continuación se describen estas variaciones con más detalle:

III.5.2.1 Sub-árboles considerando aristas (***WLE***)

Como podemos ver en la Figura (11), el kernel de sub-árboles *Weisfeiler-Lehman* (k_{WL}) (algoritmo 4) genera las etiquetas multiconjunto de cada uno de los vértices a partir de las etiquetas de los vértices en su vecindario $\mathcal{N}(v)$, es decir, los vértices que están conectados

a el.

Shervashidze y Borgwardt (2009) mencionan que es posible incluir las etiquetas de las aristas en la creación de las etiquetas multiconjunto sin embargo, no muestran cómo hacerlo.

Para implementar el kernel de sub-árboles considerando aristas (*WLE*), se modificó el paso 1 del algoritmo 4 para formar la etiqueta de un vértice v, con las etiquetas de los vértices de su vecindario $\mathcal{N}(v)$ además de las etiquetas de las aristas que lo conectan con estos vértices.

La Figura 14 ilustra una iteración del algoritmo 4 modificado para tomar en cuenta aristas. En este caso, se utilizaron los mismos grafos de entrada que en la Figura 11 agregándole 4 tipos de aristas distintos (a, b, c y d).

En esta Figura se puede ver cómo cambian las etiquetas multiconjunto para cada vértice, además, por incluir más información, estas etiquetas se vuelven más específicas y con esto, se repiten menos (lo que se puede ver al comparar las Figuras 11(c) y 14(c)).

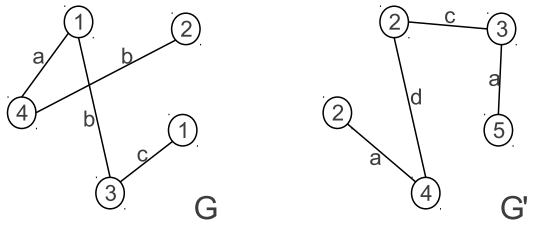
En la Figura 15 se puede ver cómo se construyen los vectores de características del ejemplo de la Figura 14 con el kernel de sub-árboles considerando aristas (*WLE*). Como se puede ver en la Figura 12, a diferencia de la versión normal (*WL*), el tamaño de los vectores es mayor, por el hecho de que se repiten menos las etiquetas.

Como consecuencia de esto, aunque más grandes, los propios vectores son más dispersos, es decir, tienen más ceros que los de su contraparte *WL* lo que pudiera mejorar el tiempo de cálculo del propio kernel *WLE*.

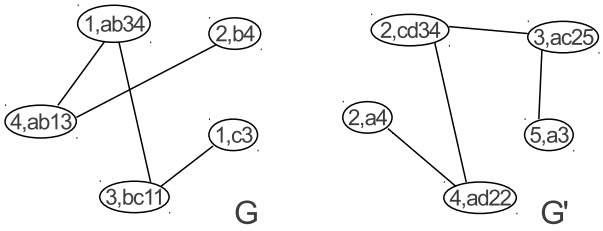
III.5.2.2 Sub-árboles sobre grafos de caminos más cortos

Como se ha mencionado, el proceso de aplicar la transformación de *Floyd-Warshall* a un grafo puede beneficiar al proceso de clasificación ya que se construye un grafo con más aristas a partir de las aristas que tiene.

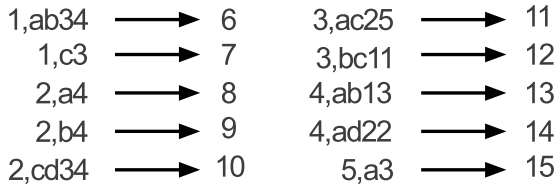
Con esto en mente, se decidió aplicar esta transformación a los grafos antes de ser procesados por los kernels de sub-árboles. A estos kernels para propósitos de referencia se



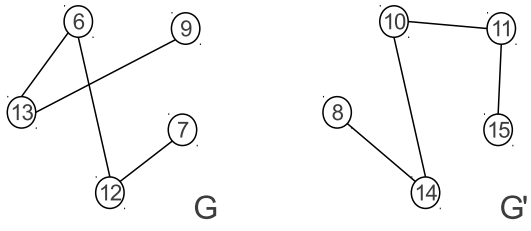
(a) Grafos de entrada G y G'.



(b) Resultado de los pasos 1 y 2: determinación de etiquetas multiconjunto y ordenamiento.



(c) Resultado del paso 3: compresión de etiquetas.



(d) Resultado del paso 4: re-etiquetado.

Figura 14. Ilustración del cálculo del kernel de sub-árboles considerando aristas WLE con h = 1 (primera iteración) para dos grafos.

$$\phi^{(1)}_{subárbolesWLE}(G) = (\ 2,1,1,1,0\ ,\ 1,1,0,1,0,0,1,1,0,0)$$
$$\phi^{(1)}_{subárbolesWLE}(G') = (\ \underbrace{0,2,1,1,1}_{\substack{\text{Conteo de etiquetas}\\ \text{de vértices originales}}}\ ,\ \underbrace{0,0,1,0,1,1,0,0,1,1}_{\substack{\text{Conteo de etiquetas}\\ \text{de vértices comprimidas}}}\ )$$

$$k^{(1)}_{subárbolesWLE}(G, G') = \langle \phi^{(1)}_{subárbolesWLE}(G), \phi^{(1)}_{subárbolesWLE}(G') \rangle = 4$$

Figura 15. Representación de los vectores de características de G y G' y cálculo del producto interno del ejemplo de la Figura 14.

les llama *WL-FW* y *WLE-FW* a los kernels de sub-árboles después de haber aplicado la transformación de *Floyd-Warshall* en su versión normal y la que toma en cuenta aristas respectivamente.

III.5.3 Aplicación de kernels sobre grafos *podados*

Es claro el impacto que tiene la *densidad* de los grafos de entrada en el tiempo de ejecución de los kernels. Este problema surge especialmente cuando se trabaja con grafos que resultan de la transformación de *Floyd-Warshall* ya que son, a exepción de los autociclos, completamente conectados y por lo tanto grafos *densos*.

La solución más simple a este problema es *quitar* o *podar* aristas de tal manera que se convierta en un grafo disperso y con esto reduzca el impacto de la *densidad* en el tiempo de ejecución.

La manera de podar estos grafos no es en forma aleatoria, en este caso se optó por extraer el árbol de esparcimiento mínimo (*MST*, por sus siglas en inglés) del grafo a tratar de tal manera que el número de aristas del propio grafo se reduce hasta $n - 1$.

III.5.3.1 Variaciones lineal, poli y rbf

Como se especificó en la ecuación 45, los kernels de *caminos más cortos* (*SP* e *iSP*) están formados por dos kernels más sencillos: k_{long} y k_{paso} para *SP* y k_{dist} y $k_{paso_{iSP}}$ para *iSP*.

A su vez, k_{paso} y $k_{paso_{iSP}}$ son formados por 3 kernels simples: $k_{vértice}$ para los vértices de origen y destino y k_{arista}, como se puede ver en las ecuaciónes 43 y 55.

Recordando, la ecuación 44 describe el kernel de *caminos más cortos* en un sentido general:

$$k_{SP}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k_{long}(e, e') k_{paso}(e, e'),$$

de esta ecuación, k_{paso} se descompone a su vez en $k_{vértice}$ y k_{arista}, como se puede ver en la ecuación 43.

Haciendo uso de las técnicas de construcción de kernels válidos, se pueden utilizar distintos kernels para la definición de estos componentes (k_{arista} y $k_{vértice}$), se pueden crear variaciones de estos kernels de *caminos más cortos* para evaluar sus resultados.

A continuación se presentan 3 definiciones de k_{arista} que se proponen como variaciones de los kernels de *caminos más cortos*.

$$k_{arista_1}(e, e') = k_{lineal}(e, e'), \tag{56}$$

$$k_{arista_2}(e, e') = k_{poli}(e, e'), \tag{57}$$

$$k_{arista_3}(e, e') = k_{RBF}(e, e'). \tag{58}$$

La razón por la cual, estos distintos kernels se aplican solo a k_{arista} es que, por lo regular el número de etiquetas para los vértices es mucho menor que el de las etiquetas de aristas. Es por esto que para $k_{vértice}$ solo se propone una sola definición (47).

De esta manera, tenemos 3 variaciones por cada una de las versiones del kernel de *caminos más cortos* (*SP* e *iSP*). Esto es, *SP-lineal*, *SP-poli* y *SP-rbf* para el primero e *iSP-lineal*, *iSP-poli* e *iSP-rbf* para el segundo.

Para los kernels que funcionan construyendo vectores de características contando ciertas secuencias dentro de los grafos como el kernel de sub-árboles *Weisfeiler-Lehman* y variaciones y el kernel *SPC*, el uso de los kernels *lineal*, *polinomial* y *RBF* se realiza de distinta manera.

Como se muestra en la Figura 52 (en el caso del kernel de sub-árboles *Weisfeiler-Lehman*), después de haber construido el vector de características que representa a cada uno de los grafos, se procede a calcular su producto interno (que es k_{lineal}). De esta manera, las otras dos variantes se implementan símplemente sustituyendo el kernel k_{lineal} por los kernels *polinomial* y *RBF*.

A continuación se muestran las 3 variantes del kernel de sub-árboles como ejemplo de la aplicación de los 3 kernels:

$$k^{(h)}_{WL-lineal}(G, G') = k_{lineal}(\phi^{(h)}_{subárbolesWL}(G), \phi^{(h)}_{subárbolesWL}(G')),$$

$$k^{(h)}_{WL-poli}(G, G') = k_{poli}(\phi^{(h)}_{subárbolesWL}(G), \phi^{(h)}_{subárbolesWL}(G')),$$

$$k^{(h)}_{WL-rbf}(G, G') = k_{RBF}(\phi^{(h)}_{subárbolesWL}(G), \phi^{(h)}_{subárbolesWL}(G')),$$

con esto se observa la aplicación de distintos kernels sobre los vectores de características de G y G'.

Capítulo IV

Experimentos y resultados

Una vez que se implementaron los métodos kernel (ver Tabla III), se procede a evaluar el desempeño de estos algoritmos tanto en tiempo de cómputo como en porcentaje de exactitud de clasificación.

Tabla III. Métodos kernel para grafos implementados.

	Tipos de kernels básicos		
Kernel	Lineal	Polinomial	RBF
Caminos más cortos	SP-lineal	SP-poli	SP-rbf
Caminos más cortos *podado*	SP-T-lineal	SP-T-poli	SP-T-rbf
Caminos más cortos invertido	iSP-lineal	iSP-poli	iSP-rbf
Caminos más cortos invertido *podado*	iSP-T-lineal	iSP-T-poli	iSP-T-rbf
Conteo de caminos más cortos	SPC-lineal	SPC-poli	SPC-rbf
Conteo de caminos más cortos *podado*	SPC-T-lineal	SPC-T-poli	SPC-T-rbf
Sub-árboles	WL-lineal	WL-poli	WL-rbf
Sub-árboles considerando aristas	WLE-lineal	WLE-poli	WLE-rbf
Sub-árboles sobre caminos más cortos	WL-FW-lineal	WL-FW-poli	WL-FW-rbf
Sub-árboles sobre caminos más cortos *podado*	WL-FW-T-lineal	WL-FW-T-poli	WL-FW-T-rbf
Sub-árboles considerando aristas sobre grafos de caminos más cortos	WLE-FW-lineal	WLE-FW-poli	WLE-FW-rbf
Sub-árboles considerando aristas sobre grafos de caminos más cortos *podado*	WLE-FW-T-lineal	WLE-FW-T-poli	WLE-FW-T-rbf

Por la forma en la que funciona cada método, su desempeño se ve afectado por tres factores: la *densidad* de los grafos, el *número de grafos* en el conjunto de datos a clasificar y el *tamaño de los grafos* de entrada (número de vértices). Es por esto que se diseñaron experimentos específicos para aislar el comportamiento de estos kernels con respecto a estas

tres variables.

Además de esto, se diseñaron experimentos con tres conjuntos de datos reales: *Mutag* (Debnath *et al.*, 1991), que es un conjunto de nitrocompuestos aromáticos y heteroaromáticos, *PTC* (Helma *et al.*, 2001), que es un conjunto de compuestos químicos suministrados a cuatro tipos de animales de prueba y *Pseudocentros* (Gramada y PE, 2006), que es un conjunto de 40 proteínas de las familias de las quinasas, transferasas y liasas carbón-carbón. En este caso el enfoque de los experimentos es en el porcentaje de exactitud de clasificación obtenido por los métodos kernel implementados.

Por último, se probaron los métodos kernel implementados sobre grafos etiquetados por medio de dos esquemas diferentes.

En el Apéndice B se describe a detalle la generación de los conjuntos de datos sintéticos así como los conjuntos de datos reales que fueron utilizados para los experimentos en este trabajo.

IV.1 Estructura de resultados

Este capítulo se divide en tres secciones principales: *Datos Reales*, *Datos Artificiales* y *Esquemas de etiquetado*.

En estas tres secciones, se prueban las variantes propuestas que utilizan el algoritmo de *caminos más cortos*, seguido de los algoritmos que emplean variaciones del kernel de *sub-árboles* en cada uno de los conjuntos de datos con los que se cuenta.

IV.2 Datos reales

En esta sección se muestran los resultados de los experimentos realizados con los conjuntos de datos *Mutag*, los cuatro grupos del conjunto de datos *PTC* (*PTC-MM*, *PTC-FM*, *PTC-MR* y *PTC-FR*) y *Pseudocentros*.

El enfoque de estos experimentos es evaluar el desempeño en porcentaje de clasificación de los métodos kernel implementados.

IV.2.1 Variantes del kernel de caminos más cortos (SP)

IV.2.1.1 SP-lineal, SP-poli, SP-rbf

En la Figura 16 se muestran los resultados de clasificación del kernel de caminos más cortos utilizando los kernels *lineal*, *polinomial* y *rbf* (*SP-lineal*, *SP-poli* y *SP-rbf*, respectivamente) en donde podemos ver que el desempeño de las tres variantes es similar.

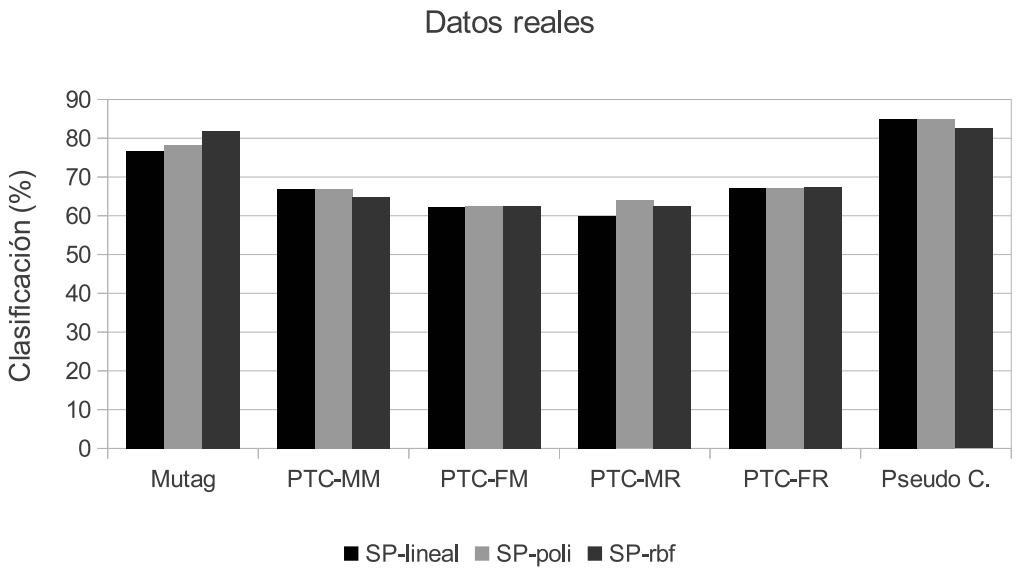


Figura 16. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*).

En este caso, la diferencia máxima entre los porcentajes de clasificación de los tres métodos es en el caso del conjunto de datos *Mutag* en donde se obtuvo un porcentaje de clasificación de 76.60% y 81.92% para *SP-lineal* y *SP-rbf*, respectivamente (diferencia de 5.32%).

Para el resto de los conjuntos de datos, los porcentajes de clasificación no estuvieron a más de 4% entre cualquiera de las tres variantes. Además, en dos ocasiones se tuvieron resultados cercanos (diferencia de 0.29%).

Estos resultados nos muestran que, para los datos reales, cualquier variante del kernel *SP* es igual de bueno.

IV.2.1.2 Kernel de caminos más cortos invertido (iSP)

Al probar el desempeño del kernel de *caminos más cortos invertido* y sus variantes (*iSP-lineal*, *iSP-poli* e *iSP-rbf*) con los conjuntos de datos reales en comparación con el original (en sus tres variantes) y analizar los resultados (Figura 17) podemos ver que en lo que se refiere a porcentaje de clasificación, ninguno de los dos superó al otro de una manera definitiva.

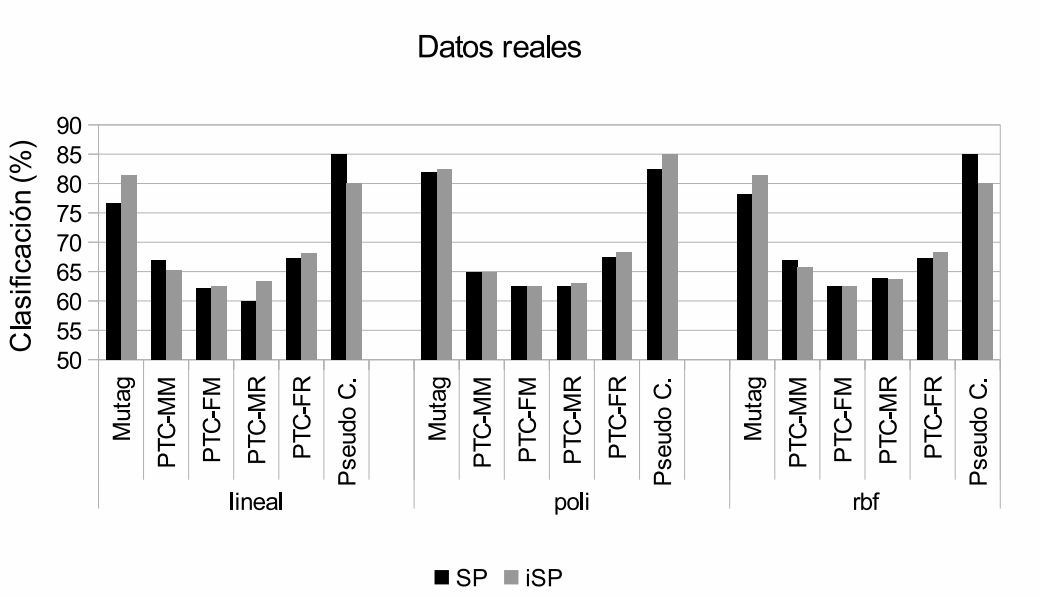


Figura 17. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos invertido y sus variantes (*iSP-lineal*, *iSP-poli* e *iSP-rbf*).

En los conjuntos de datos *Mutag* y *PTC-FR*, el kernel de *caminos más cortos invertido* supera al *kernel de caminos más cortos* original (en sus tres variantes), siendo en *Mutag* en el cual se tiene una mayor diferencia (4.7% para su versión *lineal*).

Por otro lado, para otros casos resulta tener una diferencia de hasta 5% de clasificación favorable al kernel de *caminos más cortos* original (para el conjunto de datos de *pseudo-centros*) en su versión *lineal* y *rbf*.

En el resto de los datos se tienen resultados mixtos lo cual no permite determinar como

"mejor" a ninguno sobre el otro en términos de porcentaje de clasificación en conjuntos de datos reales con los que se cuenta.

IV.2.1.3 Kernel de caminos más cortos *podado* (SP-T)

Debido a que el kernel de *caminos más cortos "podados"* reduce el número de aristas en los grafos a clasificar, se espera que el desempeño en porcentaje de clasificación disminuya a cambio de un mejor tiempo de cálculo.

Como se puede ver en la Figura 18, es claro el efecto negativo que tuvo en la clasificación pues en casi todos los casos el porcentaje de clasificación disminuyó, en el peor de los casos hasta en un 12.5% (en el conjunto de datos de *pseudocentros*) y en el mejor de los casos arrojó el mismo porcentaje de clasificación.

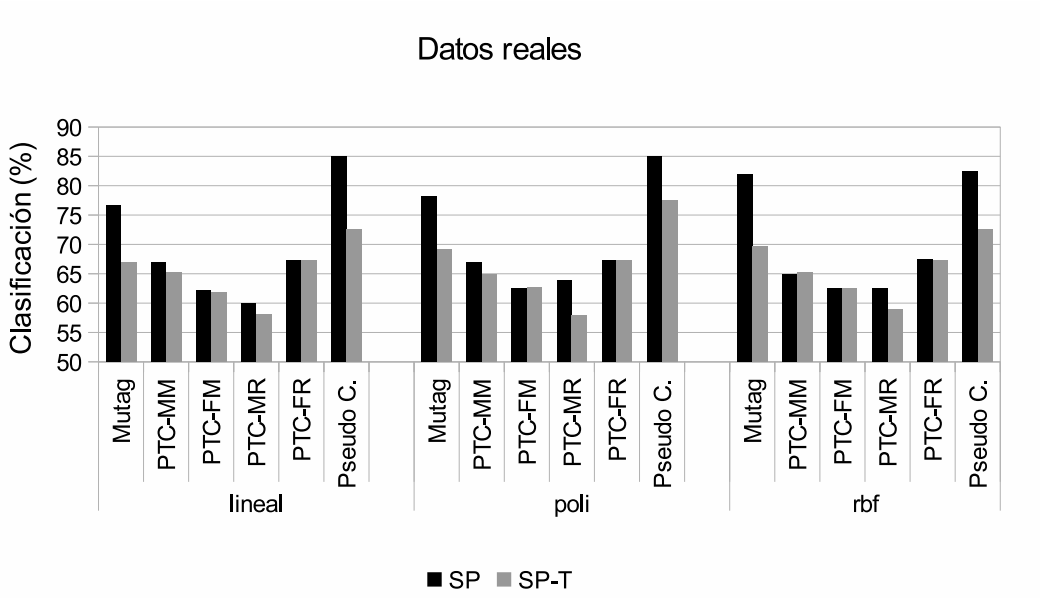


Figura 18. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos *podado* y sus variantes (*SP-T-lineal*, *SP-T-poli* y *SP-T-rbf*).

Con estos resutados se confirma el impacto negativo en porcentaje de clasificación del proceso de *podado* de los grafos de entrada, sin embargo, como se ve en los experimentos con datos sintéticos, este proceso trae consigo un mejor desempeño en tiempo de ejecución

a comparación de su versión sin *podar*.

IV.2.1.4 Kernel de caminos más cortos invertido *podado* (iSP-T)

De la misma manera que con el kernel de *caminos más cortos* podado, el kernel de *caminos más cortos invertido* podado (*iSP-T*) resulta ser inferior (Figura 19) , en porcentaje de clasificación en los conjuntos de datos con los que se cuenta, a su contraparte sin *podar* (*iSP*).

En el peor de los casos, esta variación obtiene un 10% menos en la clasificación (para el conjunto de datos *Mutag*) y en el mejor de los casos supera en menos de 1% a su contraparte (para el conjunto *PTC-FM* en las tres variantes del kernel).

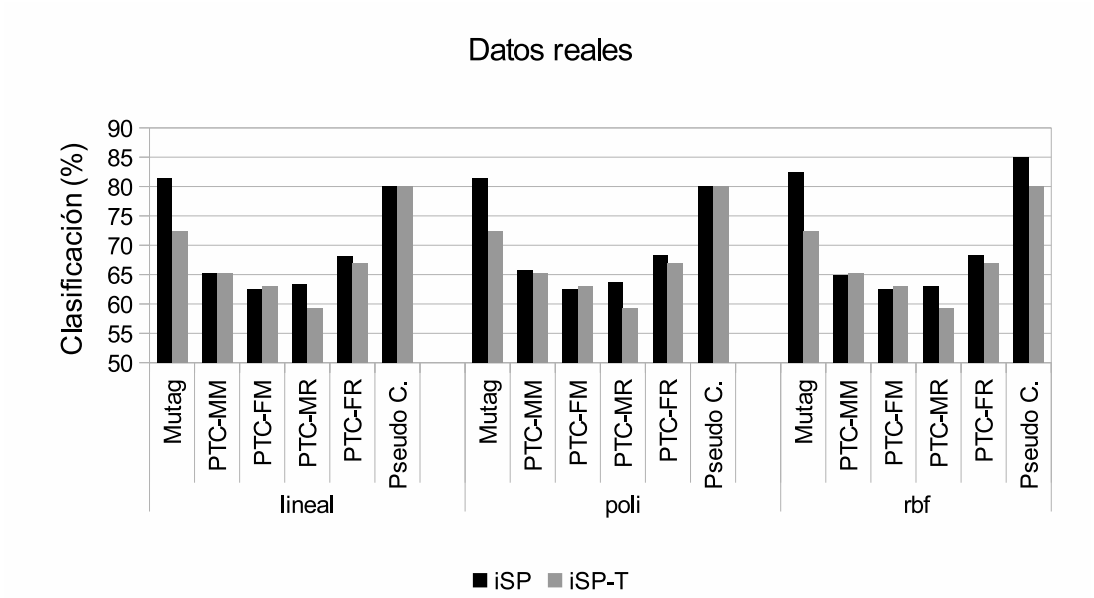


Figura 19. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de caminos más cortos invertido y sus variantes (*iSP-lineal*, *iSP-poli* e *iSP-rbf*) y kernel de caminos más cortos invertido *podado* y sus variantes(*iSP-T-lineal*, *iSP-T-poli* e *iSP-T-rbf*).

No obstante, en promedio ésta variación (*iSP-T*) tiene un impacto negativo menor al que sufre el kernel de *caminos más cortos* podado (*SP-T*), especialmente en el conjunto de datos de *pseudocentros* en donde en algunos casos no sufre impacto alguno, mientras que en la versión *SP-T* disminuye en un 12% el porcentaje de clasificación con respecto a sus

contrapartes sin *podar* respectivas.

Esta observación es importante ya que, aunque ambas variantes *podadas* (*SP-T* e *iSP-T*) sufren una disminución en el porcentaje de clasificación con respecto a sus contrapartes, la segunda lo hace en menor proporción lo que podría ser un factor determinante a su favor en ciertos casos.

IV.2.1.5 Kernel de conteo de caminos más cortos (SPC)

Los resultados de los experimentos (Figura 20) con los conjuntos de datos reales del kernel de *conteo de caminos más cortos* (*SPC*) en comparación del kernel de *caminos más cortos* original (*SP*) no muestran diferencias apreciables en la mayoría de los casos.

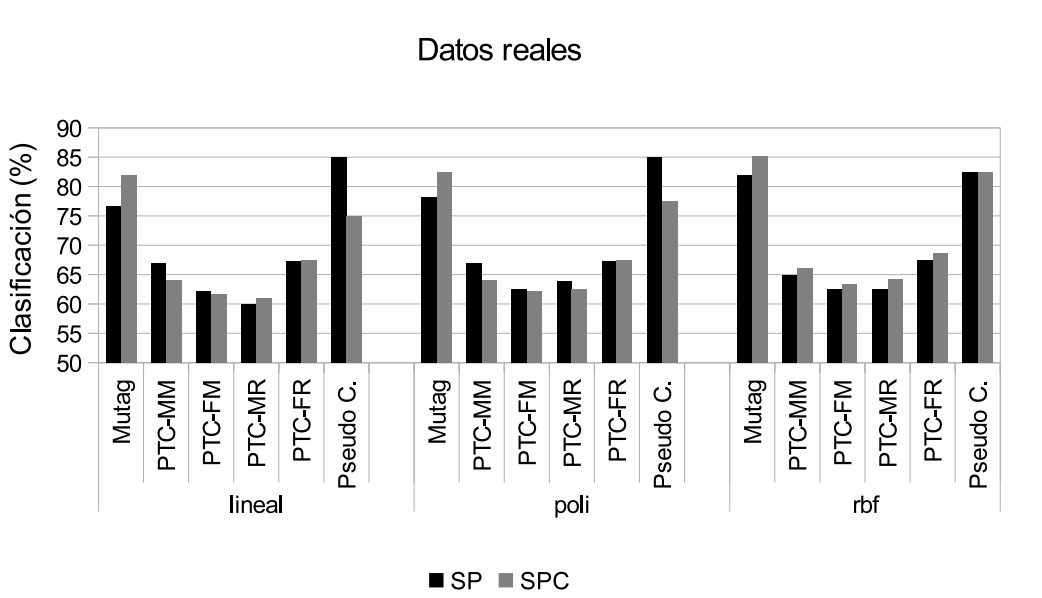


Figura 20. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de conteo de caminos más cortos y sus variantes(*SPC-lineal*, *SPC-poli* e *SPC-rbf*).

Para las variantes original y la que utiliza el kernel *polinomial* se puede observar que para algunos conjuntos de datos, como *PTC-MM* y *PTC-FM*, el kernel original (*SP*) obtiene mejores resultados que el kernel de conteo de caminos más cortos (*SPC*) mientras que para otros (*Mutag* y *PTC-FR*), se invierten los resultados, siendo *SPC* el que iguala o supera

los resultados de *SP*.

Por otro lado, para la variación que utiliza el kernel *RBF*, el kernel de *conteo de caminos más cortos* (*SPC-rbf*) resulta tener mejores resultados que el kernel de *caminos más cortos* (*SP-rbf*) en todos los conjuntos de datos. Incluso, en 4 de los 6 conjuntos de datos (*Mutag*, *PTC-FM*, *PTC-MR* y *PTC-FR*), resulta tener los mejores resultados para cualquiera de las 6 variantes en esta comparación.

IV.2.1.6 Kernel de conteo de caminos más cortos *podado* (SPC-T)

En la Figura 21 se muestran los resultados de la comparación entre el kernel de *conteo de caminos más cortos* y su versión *podada*. Como se esperaba, el porcentaje de clasificación se ve afectado por el hecho de que los grafos de entrada tienen muy pocas aristas con respecto a los grafos originales.

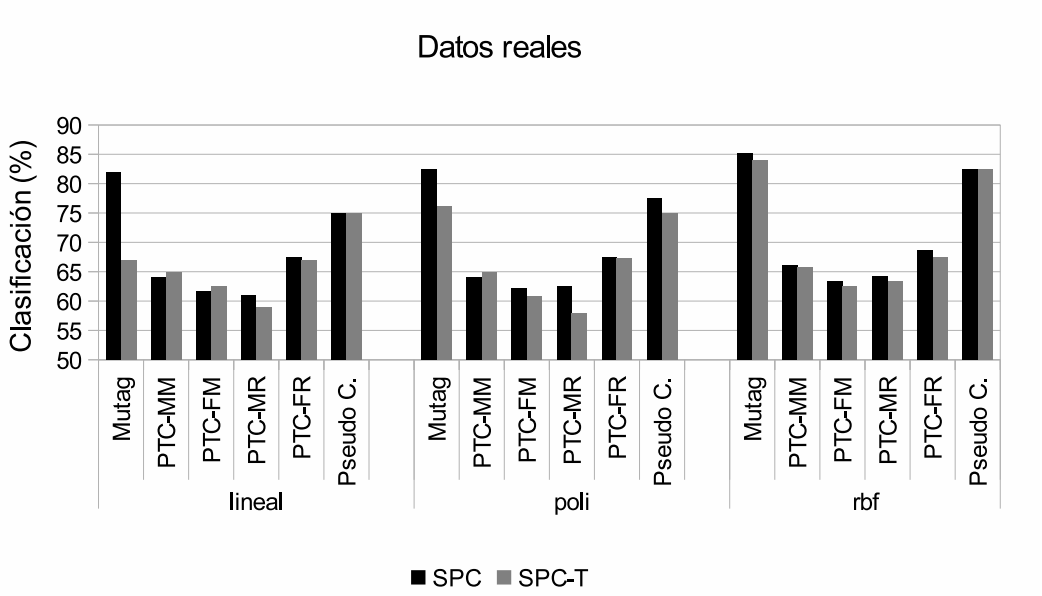


Figura 21. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de *conteo de caminos más cortos* y sus variantes (*SPC-lineal*, *SPC-poli* y *SPC-rbf*) y kernel de *conteo de caminos más cortos podado* y sus variantes (*SPC-T-lineal*, *SPC-T-poli* y *SPC-T-rbf*).

Sin embargo, con excepción a los resultados obtenidos para el conjunto de datos *Mutag* que son muy inferiores en dos de los tres casos, el resto de los resultados son comparables y

en algunos casos iguales o poco mejores que la versión con grafos completos como entradas (*SPC*).

Esto es de gran importancia ya que se puede dar el caso, bajo ciertas circunstancias, en el que sea más conveniente optar por este tipo de método en vez de su contraparte que tiene grafos densos o no *podados* como entrada inclusive a expensas de un porcentaje de clasificación menor.

IV.2.2 Variantes del kernel de subárboles (WL)

IV.2.2.1 WL-lineal, WL-poli, WL-rbf

Como se puede apreciar en la Figura 22, las dos variantes empleando los kernels *polinomial* y *RBF* (*WL-poli* y *WL-rbf*) mejoran el desempeño en porcentaje de clasificación sobre los resultados del kernel de *subárboles* original (*WL-lineal*).

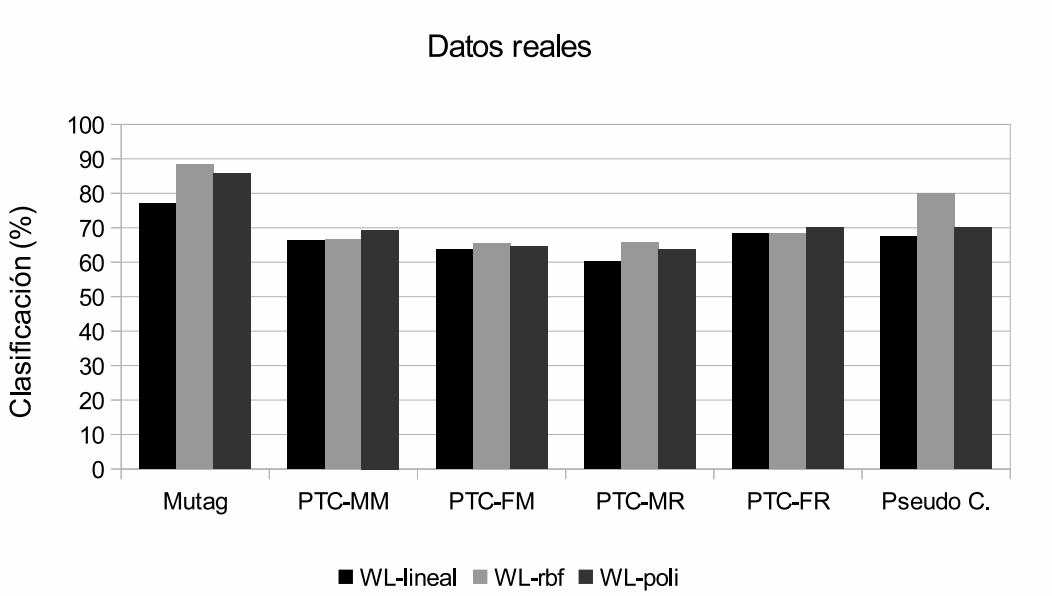


Figura 22. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de sub-árboles y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*).

Para ambos casos, los porcentajes fueron superiores para todos los conjuntos de datos. En el caso de *WL-poli*, el promedio de clasificación fue 3.34% mayor al original siendo el resultado del conjunto de datos *Mutag* el que tuvo una diferencia más grande con 8.51%.

La variante que utiliza el kernel *RBF* (*WL-rbf*) tuvo el mejor desempeño con respecto a *WL-lineal* teniendo, en promedio, un porcentaje de clasificación superior en 5.2%.

IV.2.2.2 Kernel de subárboles considerando aristas (WLE)

Para la versión del kernel de *subárboles* que toma en cuenta las aristas (*WLE*), el resultado de la comparación de porcentajes de clasificación con su contraparte original (*WL*) no mostró diferencias.

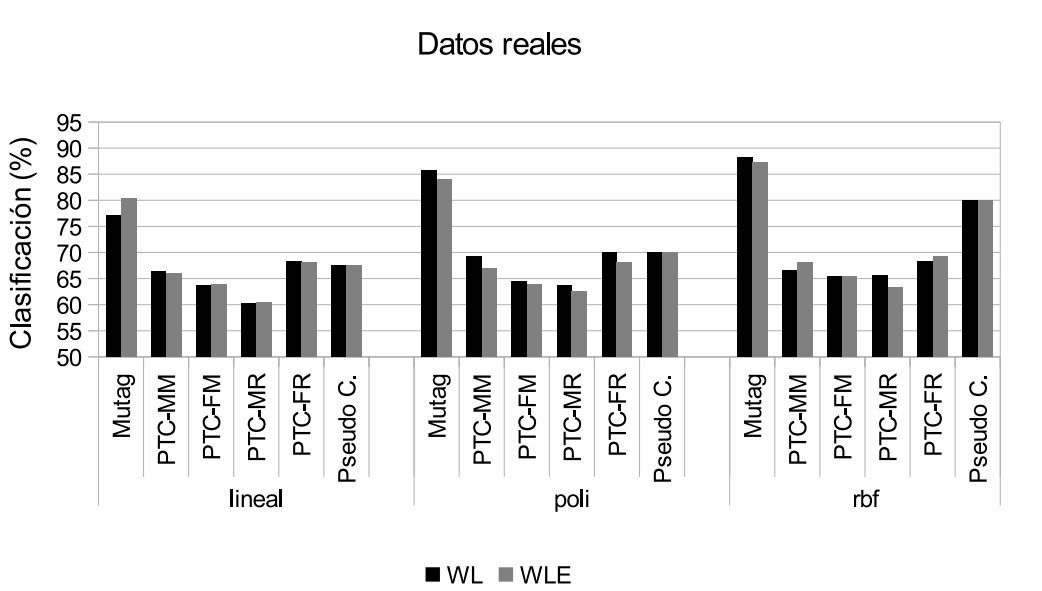


Figura 23. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de sub-árboles y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de sub-árboles considerando aristas y sus variantes (*WLE-lineal*, *WLE-poli* y *WLE-rbf*).

Por un lado, la variante sobre el original (*WLE-lineal*) obtiene virtualmente los mismos resultados que su contraparte (*WL*) ya que solo para *Mutag* obtiene un porcentaje menor que él por un poco más de 3%.

Para las otras dos variantes (*WLE-poli* y *WLE-rbf*) los resultados son aún menos distantes entre si. *WL* superó por 1.28% y 0.17% como promedio sobre todos los conjuntos de datos a *WLE* en las versiones que utilizan los kernels *polinomial* y *RBF*, respectivamente.

IV.2.2.3 Kernel de subárboles sobre el grafo de caminos más cortos (WL-FW)

El kernel de *subárboles* sobre el grafo de caminos más cortos se implementó pensando en las características extras que ofrece el grafo de caminos más cortos al ser un grafo denso, sin embargo, de acuerdo con la Figura 24 se puede observar que estas características no afectaron el desempeño de clasificación del original de manera alguna.

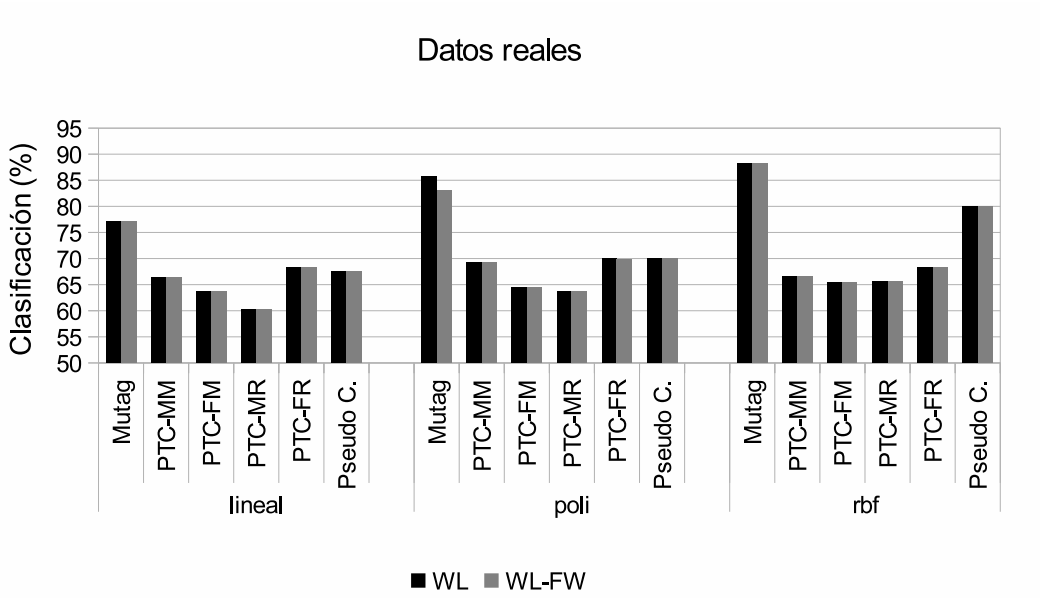


Figura 24. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de *sub-árboles* y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variantes (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*).

En este caso, todos los resultados con excepción de dos son iguales. Solamente para la variación que emplea el kernel *polinomial*, el kernel de *subárboles sobre el grafo de caminos más cortos* (*WL*) tuvo resultados negativos con respecto a su contraparte (*WL-FW*), siendo en el conjunto de datos *Mutag* en donde tuvo la diferencia máxima con 2.7% menos.

IV.2.2.4 Kernel de subárboles considerando aristas sobre el grafo de caminos más cortos (WLE-FW)

Los resultados de la comparación entre el kernel de *subárboles sobre el grafo de caminos más cortos* considerando aristas y su contraparte sobre los grafos iguales (Figura 25) muestran

que ambos métodos (*WLE* y *WLE-FW*) son comparables.

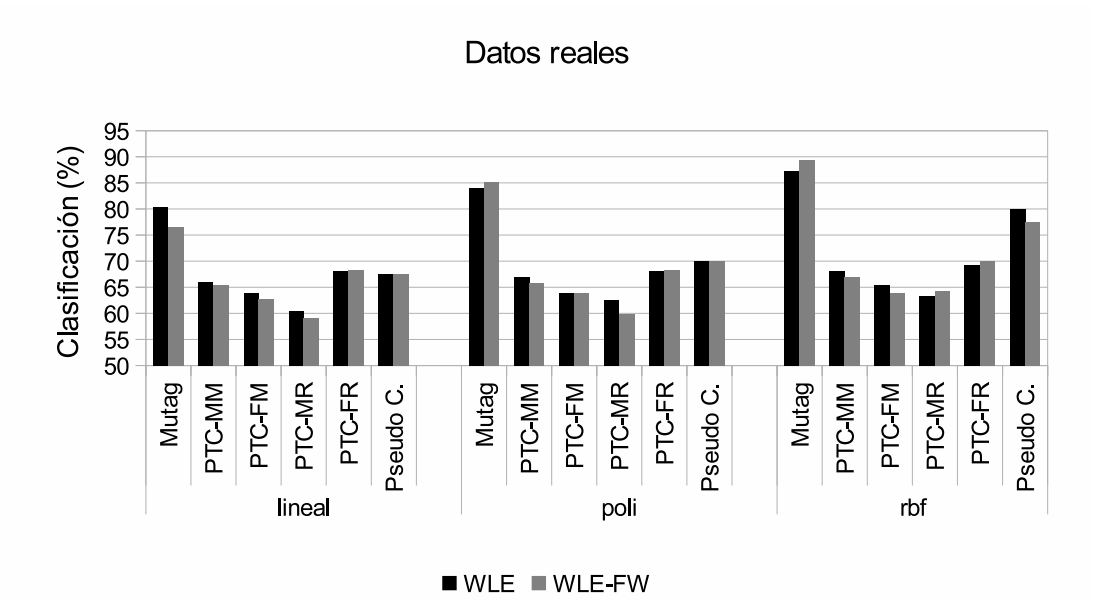


Figura 25. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de *sub-árboles* considerando aristas y sus variantes (*WLE-lineal*, *WLE-poli* y *WLE-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos considerando aristas y sus variantes (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*).

Las diferencias más grandes se dieron para el conjunto de datos de *Mutag* en donde *WLE-lineal* tuvo un porcentaje de clasificación mayor que su contraparte *WLE-FW-lineal* por 3.7%. Por otro lado, para el mismo conjunto, las variantes sobre grafos de caminos más cortos *WLE-FW-poli* y *WLE-FW-rbf* tuvieron mejores resultados con 1.06% y 2.12% mayores que sus contrapartes *WLE-poli* y *WLE-rbf*, respectivamente.

Para el resto de los conjuntos de datos, las diferencias de los resultados entre las variantes sobre grafos de caminos más cortos y su contraparte no son lo suficientemente grandes para poder determinar un claro "ganador", sin embargo, podemos observar que la mayoría de los resultados son menores, aunque no por mucho, que las versiónes aplicadas sobre los grafos originales.

IV.2.2.5 Kernel de subárboles sobre el grafo de caminos más cortos *podado* (WL-FW-T)

Al igual que la comparación de la Figura 24, todos los resultados de la comparación entre el kernel de *subárboles* sobre caminos más cortos (WL-FW) y su contraparte aplicado sobre grafos de caminos más cortos *podado* (WL-FW-T) resultaron iguales en todos los conjuntos de datos y para las tres variantes de kernels (*lineal*, *polinomial* y *RBF*).

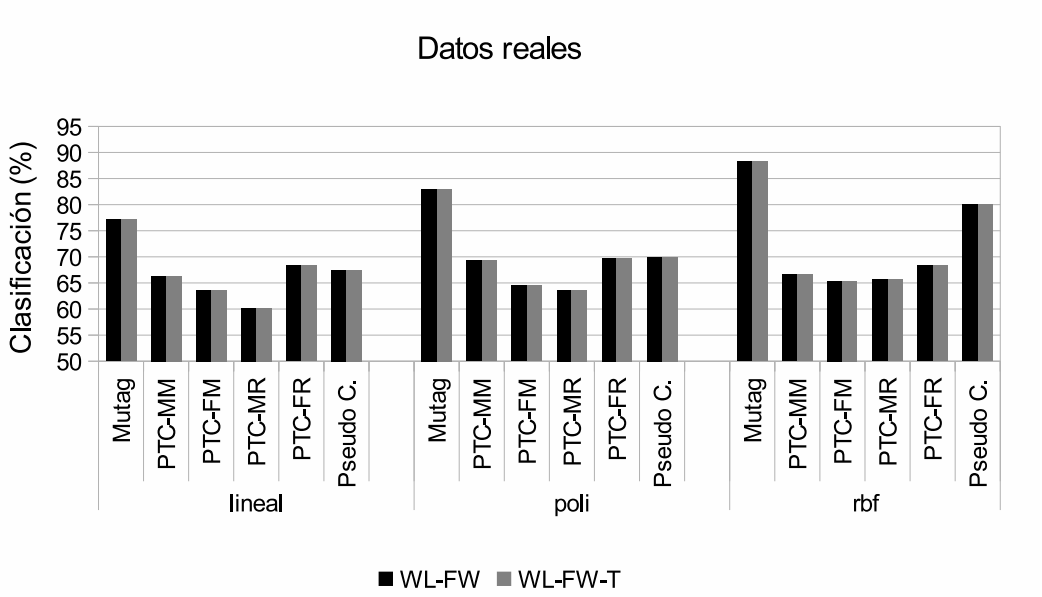


Figura 26. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variantes (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podado* y sus variantes (*WL-FW-T-lineal*, *WL-FW-T-poli* y *WL-FW-T-rbf*).

En la Figura 26, se pueden ver cómo los resultados son idénticos para todas las comparaciones. Esto resulta valioso ya que, si existe alguna mejora en cualquiera de estas dos variantes del original (*WL*), los resultados de clasificación no se verán perjudicados en gran manera e incluso serán mejorados en algunos casos (sobre todo para los que utilizan el kernel *RBF*).

IV.2.2.6 Kernel de subárboles tomando en cuenta las aristas sobre el grafo de caminos más cortos *podado* (WLE-FW-T)

En este caso, al igual que la comparación entre *WLE* y *WLE-FW*, la mayoría de los resultados son muy parecidos con excepción de cinco que están por arriba del 2% de diferencia entre si.

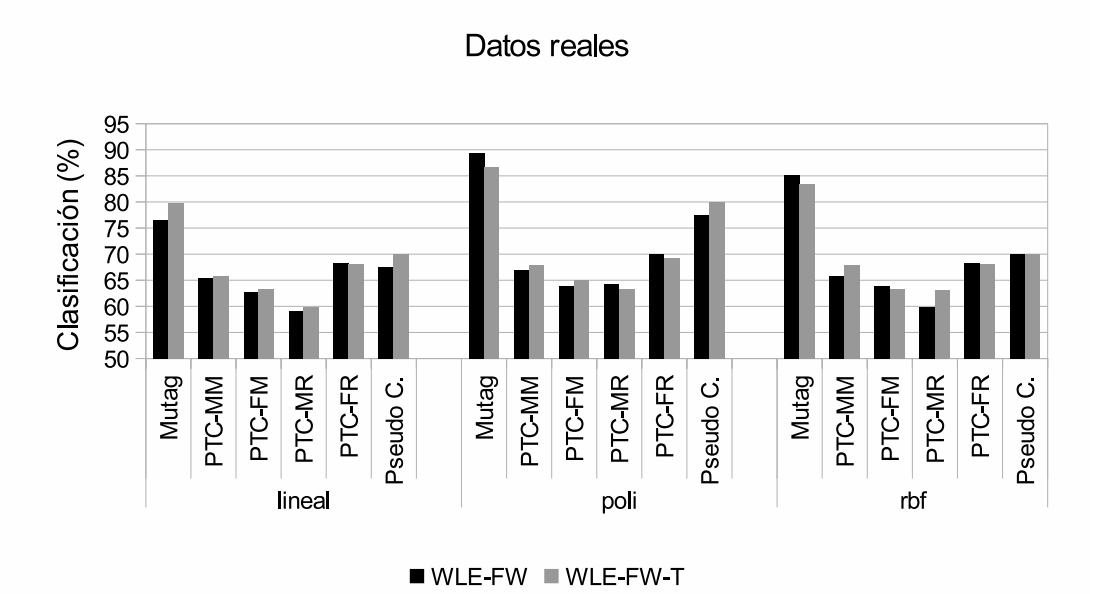


Figura 27. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernel de *sub-árboles* sobre grafo de caminos más cortos considerando aristas y sus variantes (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podado* considerando aristas y sus variantes (*WLE-FW-T-lineal*, *WLE-FW-T-poli* y *WLE-FW-T-rbf*).

La Figura 27 muestra resultados favorables para la variación *podada* (*WLE-FW-T*) solo para la versión que utiliza el kernel *lineal* (*WLE-FW-T-lineal*) ya que en cinco de los seis conjuntos de datos obtuvo un mejor porcentaje de clasificación y en el que no, es básicamente igual a su contraparte (*WLE-FW-lineal*) pues se ve superado por 0.28%.

Para las otras dos variantes (*polinomial* y *RBF*), los resultados no son suficientes para considerar a una mejor que la otra. Esto debido a que en ambas variantes los resultados son favorables para una, en cierto conjunto de datos pero desfavorable en otros, sin tener un número mayoritario como en el caso de la variación que utiliza el kernel *lineal*.

Tabla IV. Resultados de clasificación (%) de la totalidad de los métodos kernel evaluados para los conjuntos de datos reales: Mutag, PTC-MM, PTC-FM, PTC-MR, PTC-FR y Pseudocentros.

Kernel	Mutag	PTC-MM	PTC-FM	PTC-MR	PTC-FR	Pseudo C.
iSP-lineal	81.383	65.179	62.464	63.372	68.091	80
iSP-poli	81.383	65.774	62.464	63.663	68.376	80
iSP-rbf	82.447	64.881	62.464	63.081	68.376	**85**
iSP-T-lineal	72.34	65.179	63.037	59.302	66.952	80
iSP-T-poli	72.34	65.179	63.037	59.302	66.952	80
iSP-T-rbf	72.34	65.179	63.037	59.302	66.952	80
SP-lineal	76.596	66.964	62.178	59.884	67.236	**85**
SP-poli	78.191	66.964	62.464	63.953	67.236	**85**
SP-rbf	81.915	64.881	62.464	62.5	67.521	82.5
SP-T-lineal	67.021	65.179	61.891	58.14	67.236	72.5
SP-T-poli	69.681	65.179	62.464	59.012	67.236	72.5
SP-T-rbf	69.149	64.881	62.751	57.849	67.236	77.5
SPC-lineal	81.915	63.988	61.605	61.047	67.521	75
SPC-poli	82.447	63.988	62.178	62.5	67.521	77.5
SPC-rbf	85.106	66.071	63.324	64.244	68.661	82.5
SPC-T-lineal	67.021	64.881	62.464	59.012	66.952	75
SPC-T-poli	76.064	64.881	62.464	57.849	67.236	75
SPC-T-rbf	84.043	65.774	60.745	63.372	67.521	82.5
WL-FW-lineal	77.128	66.369	63.61	60.174	68.376	67.5
WL-FW-poli	82.979	**69.345**	64.47	63.663	69.801	70
WL-FW-rbf	88.298	66.667	**65.33**	**65.698**	68.376	80
WL-FW-T-lineal	77.128	66.369	63.61	60.174	68.376	67.5
WL-FW-T-poli	82.979	**69.345**	64.47	63.663	69.801	70
WL-FW-T-rbf	88.298	66.667	**65.33**	**65.698**	68.376	80
WL-lineal	77.128	66.369	63.61	60.174	68.376	67.5
WL-poli	85.638	**69.345**	64.47	63.663	**70.085**	70
WL-rbf	88.298	66.667	**65.33**	**65.698**	68.376	80
WLE-FW-lineal	76.596	65.476	62.751	59.012	68.376	67.5
WLE-FW-poli	85.106	65.774	63.897	59.884	68.376	70
WLE-FW-rbf	**89.362**	66.964	63.897	64.244	**70.085**	77.5
WLE-FW-T-lineal	79.787	65.774	63.324	59.884	68.091	70
WLE-FW-T-poli	83.511	67.857	63.324	63.081	68.091	70
WLE-FW-T-rbf	86.702	67.857	65.043	63.372	69.231	80
WLE-lineal	80.319	66.071	63.897	60.465	68.091	67.5
WLE-poli	84.043	66.964	63.897	62.5	68.091	70
WLE-rbf	87.234	68.155	**65.33**	63.372	69.231	80

IV.3 Datos sintéticos

En esta sección se presentan los resultados de los experimentos realizados con datos sintéticos diseñados y generados con el propósito de evaluar el comportamiento en tiempo de ejecución de los métodos kernels con respecto a tres variables principales de los datos a tratar: la *densidad* de los grafos, el *número* de grafos y el *tamaño* de los mismos. Estos tres conjuntos de datos se describen con más detalle en la sección B.4.

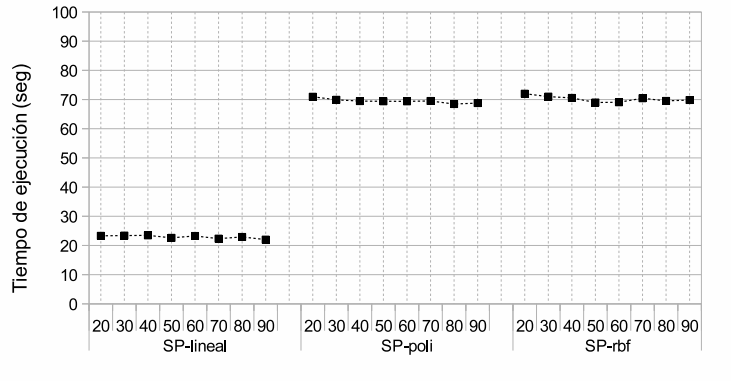
IV.3.1 Variantes del kernel de caminos más cortos (SP)

IV.3.1.1 SP-lineal, SP-poli, SP-rbf

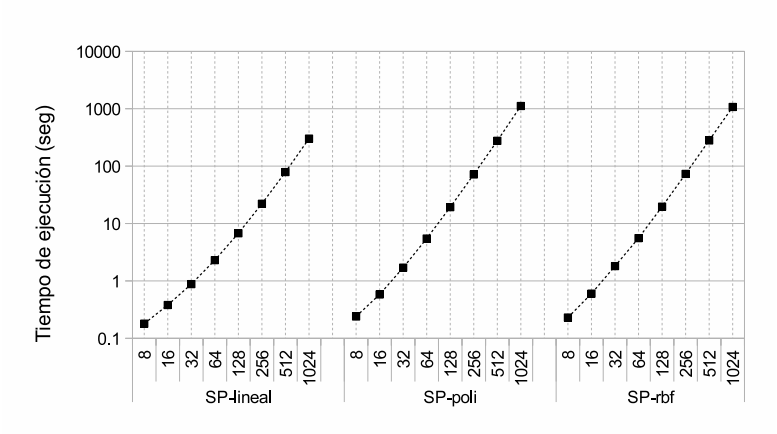
En la comparación de las variantes del kernel de *caminos más cortos* que utilizan los kernels *lineal*, *polinomial* y *RBF* (Figura 28) podemos ver, de acuerdo a la Figura 28(a), que la tendencia de crecimiento con respecto a la *densidad* de los grafos es básicamente la misma para los tres (*SP-lineal*, *SP-poli* y *SP-rbf*). No obstante, es obvio ver la diferencia en tiempo de ejecución entre el kernel *SP-lineal* y los otros dos, en donde el primero promedia un tiempo de 22.91 segundos mientras que *SP-poli* y *SP-rbf* promedian 70.18 y 69.49 segundos, respectivamente.

Cuando lo que varía es el número de grafos en el conjunto (Figura 28(b)), los tres kernels tuvieron un crecimiento exponencial aunque para su versión *lineal*, la razón de cambio fue mucho menor que la de las variantes que utilizan el kernel *polinomial* y *rbf*.

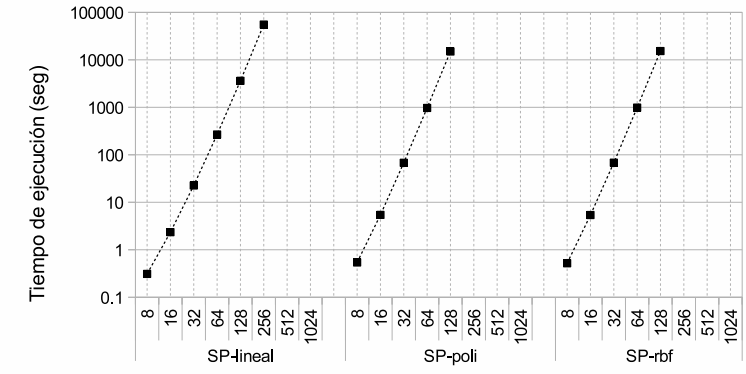
Por último, en la Figura 28(c) se muestra el comportamiento del tiempo de cálculo de los kernels con respecto al cambio de tamaño de los grafos en el conjunto de datos a tratar en donde se puede ver que este factor afecta mucho más que los otros dos. En este caso, ni siquiera se pudieron terminar las pruebas pues duraron más de dos días ejecutándose. Sin embargo, de igual manera que las otras dos pruebas (*densidad* y *número* de grafos), el kernel que menos efecto tuvo es la variación que utiliza el kernel *SP-lineal* ya que teniendo una razón de cambio mucho menor a las otras dos (*SP-poli* y *SP-rbf*).



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de los grafos **n**.

Figura 28. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*).

IV.3.2 Variantes del kernel de subárboles (WL)

IV.3.2.1 WL-lineal, WL-poli, WL-rbf

Al probar las variantes de kernels *lineal*, *poli* y *rbf* con el kernel de *subárboles* (*WL*) se puede observar (Figura 29) que el comportamiento del tiempo de cálculo con respecto a los tres parámetros: *densidad*, *número* y *tamaño* de los grafos (Figuras 29(a), 29(b) y 29(c), respectivamente) es similar en los tres casos.

Esto resulta una ventaja ya que, en caso de ofrecer mejorías en porcentaje de clasificación, se puede implementar cualquiera de las tres variantes sin penalización en tiempo de cálculo.

IV.4 Esquemas de etiquetado

En esta sección se presentan parte de los resultados de los experimentos realizados con conjuntos de datos que contienen grafos generados y etiquetados de acuerdo a uno de dos esquemas de acuerdo al recorrido de los árboles de esparcimiento mínimo de cada uno de los grafos.

En la sección B.4.4 describe con más detalle el proceso de etiquetado de los grafos en dichos conjuntos.

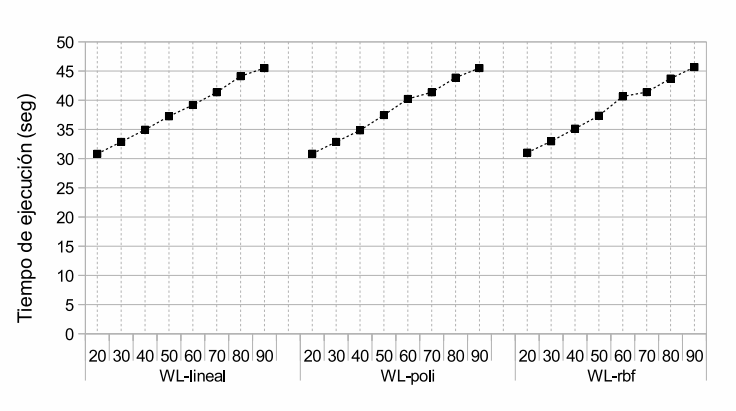
IV.4.1 Variantes del kernel de caminos más cortos (SP)

IV.4.1.1 Kernel de caminos más cortos variantes SP-lineal, SP-poli y SP-rbf

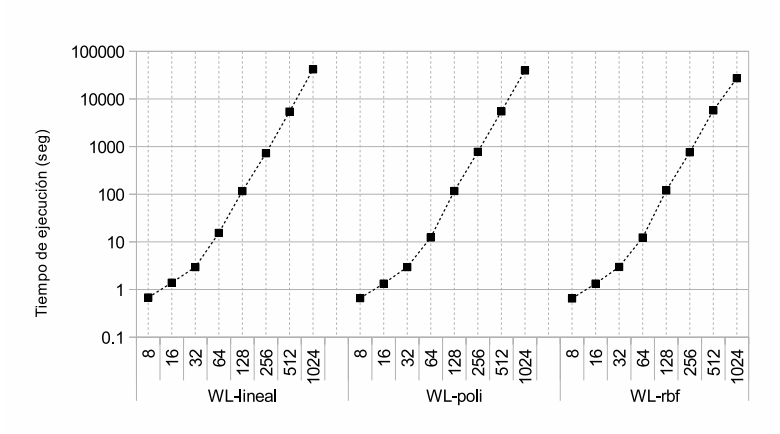
La Figura 30 muestra el desempeño de clasificación de grafos de acuerdo a esquemas de clasificación para el kernel de *caminos más cortos* en sus tres variantes (*SP-lineal*, *SP-poli* y *SP-rbf*).

Se puede observar cómo para el primer grupo (*N grande*), la variante *SP-rbf* supera en los cinco tipos de grafos procesados en donde las topologías de *árbol* y *estrella* tienen los mejores resultados con 96.17% y 90.67%, respectivamente.

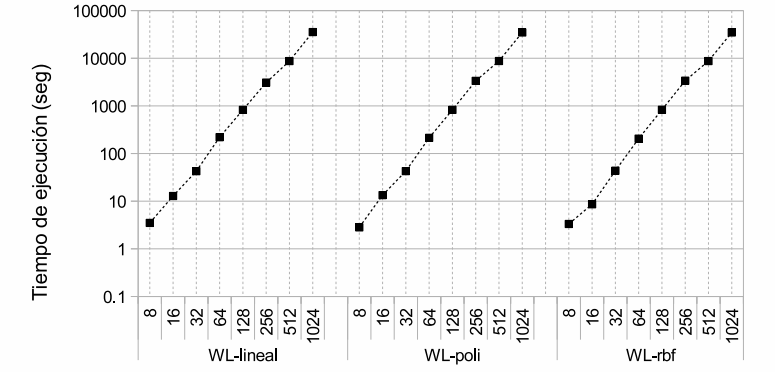
Como en la mayoría de los casos, el segundo grupo es el que sufrió un descenso general



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 29. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de sub-árboles y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*).

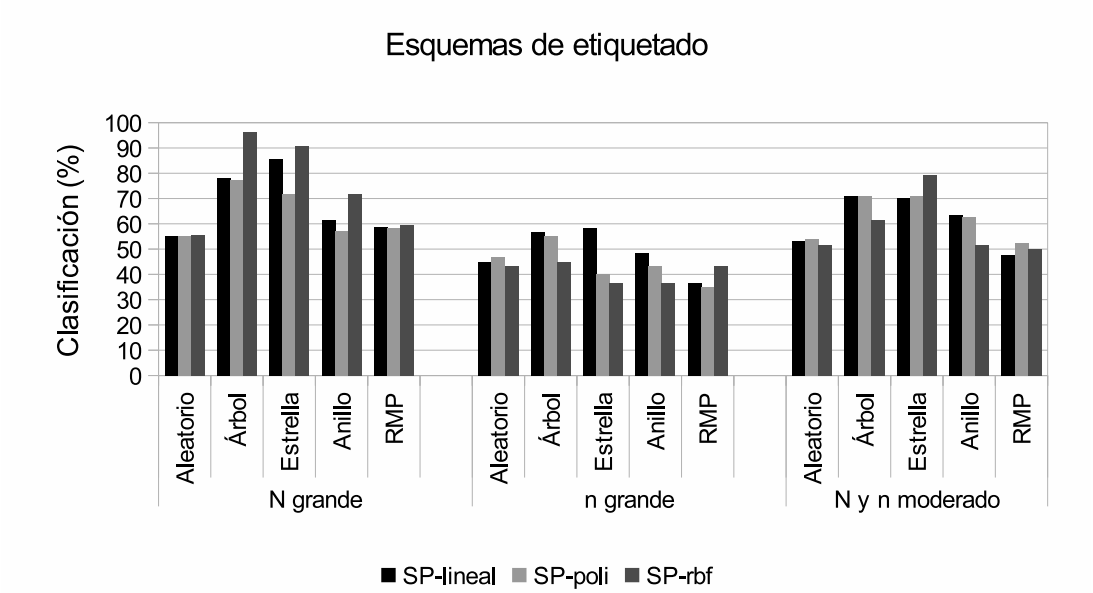


Figura 30. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

del porcentaje de clasificación con respecto a los otros dos grupos con porcentajes promedio de 49%, 44% y 41% para los kernels *SP-lineal*, *SP-poli* y *SP-rbf*, respectivamente.

Por último, en el caso del tercer grupo, se puede ver como el kernel *SP-poli* tiene resultados muy similares al original (*SP-lineal*), mientras que la variante *SP-rbf* es inferior a ambos en la mayoría de los casos con excepción de cuando se manejan grafos con topología estrella, en donde con 79.17% es superior a los otros dos en casi 10%.

IV.4.2 Variantes del kernel de subárboles (WL)

IV.4.2.1 Kernel de subárboles variantes WL-lineal, WL-poli y WL-rbf

La Figura 31 muestra el desempeño de clasificación de grafos por esquemas de clasificación para el kernel de *subárboles* en sus tres variantes (*WL-lineal*, *WL-poli* y *WL-rbf*).

Primeramente, se puede observar que en la mayoría de los casos, las tres variantes tuvieron resultados similares. Además, se puede observar cómo solo en dos ocasiones el kernel original *WL-lineal* fue superado en porcentaje de clasificación considerablemente

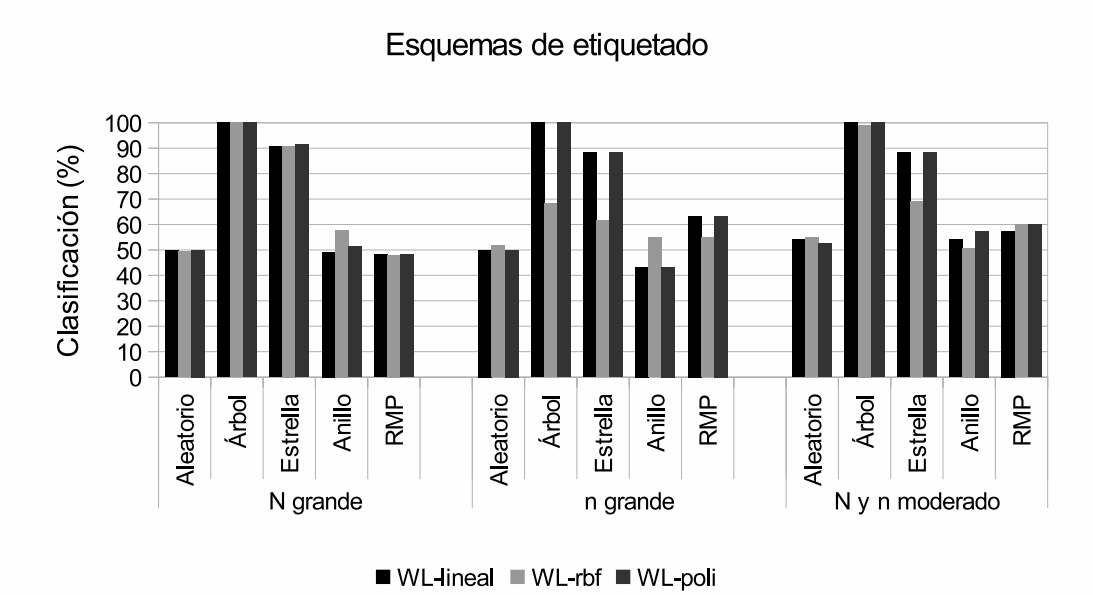


Figura 31. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de sub-árboles y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

por alguno de las dos variantes adicionales (por *WL-rbf* en todos los casos).

El kernel *WL-rbf* tuvo resultados comparables o poco mejores que los otros dos (*WL-lineal* y *WL-poli*) en 10 de los 15 casos. Sin embargo, en el resto se vió inferior a estos kernels por un rango considerable.

Por otro lado, *WL-poli* se mantuvo muy cerca de los resultados de *WL-lineal* aunque en ninguno de los casos lo superó por un margen significativo.

Capítulo V

Mejores resultados

Ya que para cada uno de los experimentos realizados, existen distintos métodos kernel con mejor desempeño y debido a que en el trabajo se comparan los métodos kernel de *caminos más cortos* y de *subárboles* con sus respectivas variantes, se comparan los dos kernels con mejor desempeño (uno de cada *familia*) en cada experimento.

La Tabla V muestra los métodos kernel de las dos *familias* que obtuvieron mejores resultados en cada uno de los experimentos realizados. Estos kernels fueron seleccionados según el experimento y en base a ciertos criterios que se explicarán a lo largo de esta sección.

Además de mostrar la comparación de los métodos kernel con mejor desempeño de cada uno de los experimentos, se muestra la comparación de dichos kernels con respecto al resto de los experimentos para así tener una mejor idea de las fortalezas y debilidades de cada uno de ellos. De esta manera, se puede saber qué es lo que se sacrifica cuando se selecciona alguno a la hora de tomar en cuenta el desempeño del método en un experimento específico.

En las figuras correspondientes a los experimentos con datos reales (*Mutag*, *PTC* y *Pseudocentros*), también se incluyen los porcentajes de clasificación de los kernels que han tenido mejores resultados en la literatura.

Para el conjunto de datos *Mutag*, se muestra el resultado obtenido con el kernel de *descubrimiento de patrones* de Kramer y Readt (2001) y para los resultados del conjunto de datos PTC, se muestran los porcentajes de clasificación obtenidos por los kernels *Tanimoto* (para *PTC-MM*, *PTC-MR* y *PTC-FR*) y *MinMax* (para *PTC-FM*), ambos de Swamidass *et al.* (2005).

Tabla V. Resumen de métodos kernel con mejores resultados por tipo de experimento para cada tipo principal: Caminos más cortos y Sub-árboles.

Experimento	***Caminos más cortos***	***Sub-árboles***
Datos reales	SPC-rbf	WL-FW-rbf
Densidad	SPC-T-poli	WL-FW-rbf
Número de grafos	SPC-T-poli	WLE-FW-rbf
Tamaño de grafos	SPC-T-poli	WLE-rbf
Esquemas de etiquetado	SPC-poli	WLE-FW-T-lineal

V.1 Conjunto de datos reales

En el experimento en el que se evalua el desempeño de clasificación sobre los conjuntos reales con los que se cuenta (*Mutag*, *PTC* y *Pseudocentros*), los métodos kernel que resultaron con mejor desempeño son: el kernel de conteo de caminos más cortos variante RBF (*SPC-rbf*) y el kernel de subárboles sobre grafo de caminos más cortos variante RBF (*WL-FW-rbf*).

Ya que no existe ningún kernel que sea el que tiene los mejores porcentajes de clasificación para todos los conjuntos de datos, para seleccionar los *mejores* kernels de *caminos más cortos* y *subárboles*, se promediaron sus respectivos porcentajes de clasificación de los cinco conjuntos de datos y posteriormente se seleccionó el de mejor porcentaje promedio de los dos tipos principales.

V.1.0.2 Kernel de caminos más cortos *SP* vs kernel de subárboles *WL*

Como se puede ver en la Figura 32, el kernel basado en *subárboles WL-FW-rbf* resultó tener mejor desempeño de clasificación en la cuatro de los seis conjuntos de datos, sin embargo, la diferencia entre ambos no es suficiente como para declararlo mejor que *SPC-rbf*. En promedio *WL-FW-rbf* supera al kernel *SPC-rbf* solo por 0.74% y la diferencia máxima a su favor es de 3.1% (*Mutag*). Por otro lado, la diferencia máxima a favor de *SPC-rbf* es de 2.5% en el conjunto *Pseudocentros*.

En el caso de los experimentos con los conjuntos de datos sintéticos (Figura 33), es decir, con respecto a la *densidad*, *número* y *tamaño*, el kernel *WL-FW-rbf* resulta tener un

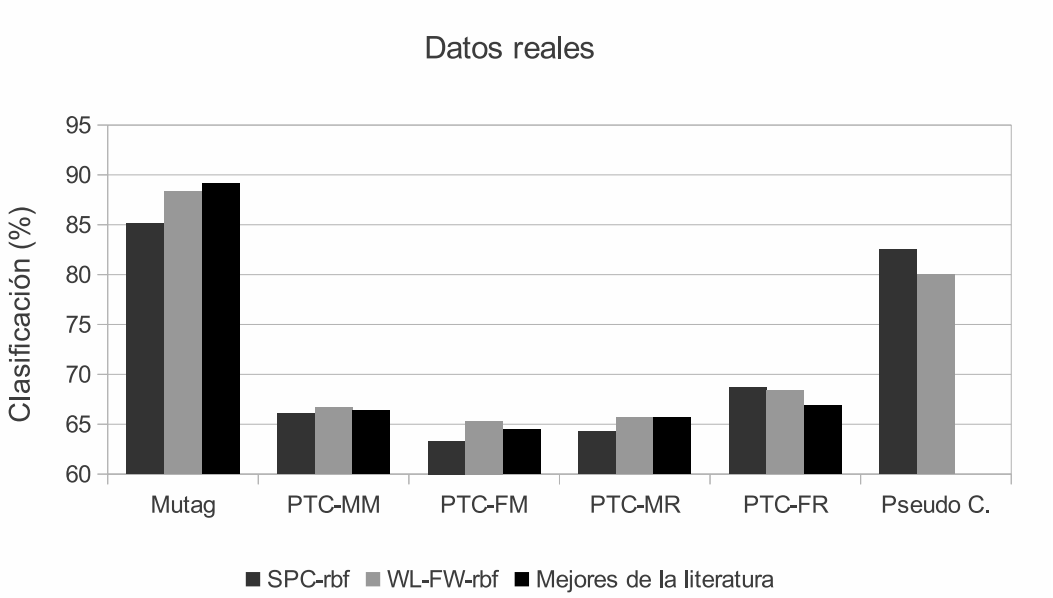
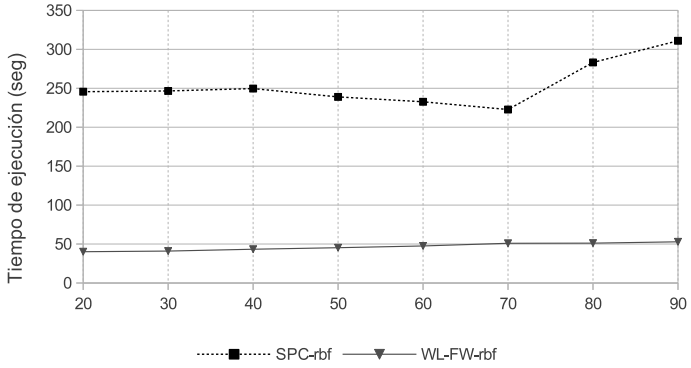


Figura 32. Porcentaje de clasificación para *Mutag*, *PTC* y *Pseudocentros* de kernels con mejor desempeño con datos reales. Kernels implementados: conteo de caminos más cortos variante RBF (*SPC-rbf*) y subárboles sobre grafos de caminos más cortos variante RBF (*WL-FW-rbf*). Mejores kernels en la literatura: kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.

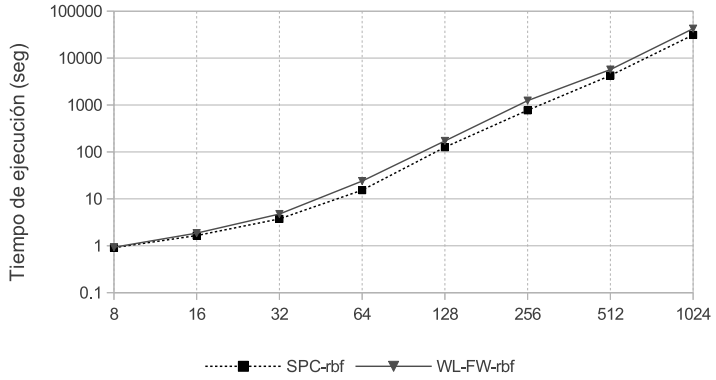
desempeño similar o superior a su contraparte *SPC-rbf*.

Solo en el caso del experimento en el cual se varía el *número* de grafos (Figura 33(b)) de entrada el kernel de la familia de *caminos más cortos* resulta comparable ya que se comporta de la misma manera que *WL-FW-rbf*. Para los otros dos casos, el kernel *WL-FW-rbf* resulta ser superior, especialmente en el caso de la Figura 33(c) en donde la diferencia es tan grande que el kernel *SPC-rbf* fue incapaz de terminar en menos de 48 horas en los últimos tres tamaños (256, 512 y 1024 grafos) casos mientras que *WL-FW-rbf* falló en el último.

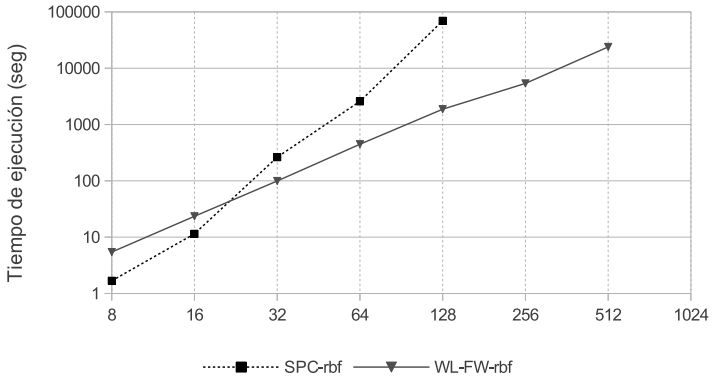
Para el experimento en el que se clasifican esquemas de etiquetado, el kernel basado en *caminos más cortos* tiene un mejor desempeño en todos los casos con excepción de dos en donde es superado en no más de 4% (esto se puede ver en la Figura 34). En general, *SPC-rbf* es superior que su contraparte *WL-FW-rbf* ya que en algunos casos la diferencia es de más de 15%. Las diferencias de porcentajes de clasificación máximas de éste kernel es de 29.1% y 31.8%.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 33. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernels con mejor desempeño para datos reales: conteo de caminos más cortos variante RBF (*SPC-rbf*) y subárboles sobre grafo de caminos más cortos variante RBF (*WL-FW-rbf*).

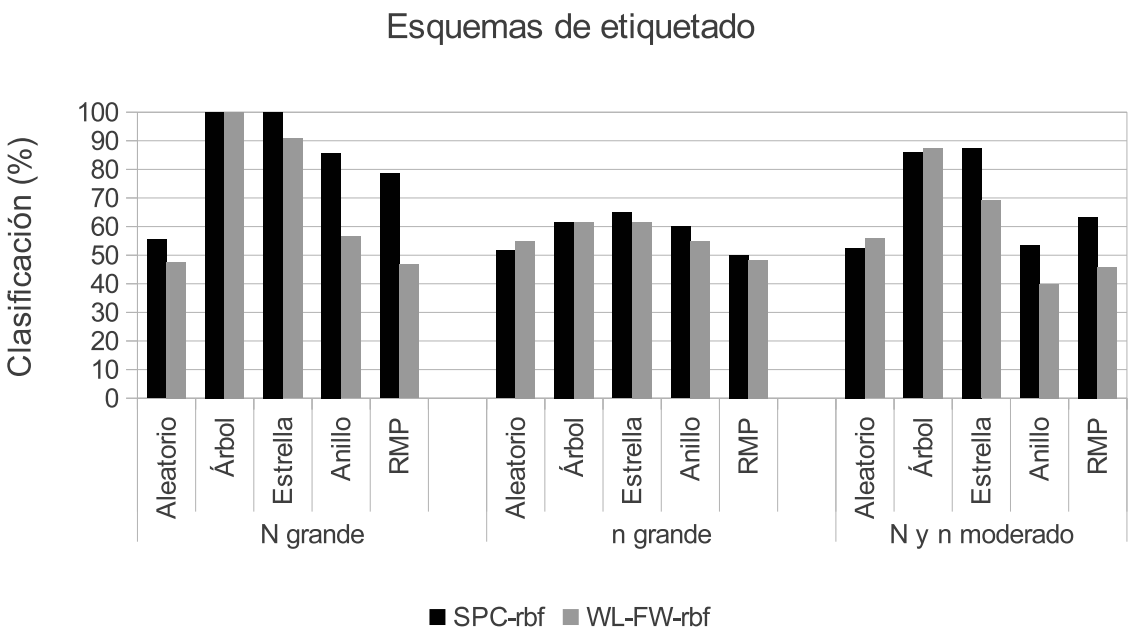


Figura 34. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernels con mejor desempeño para datos reales: conteo de caminos más cortos variante RBF (*SPC-rbf*) y subárboles sobre grafo de caminos más cortos variante RBF (*WL-FW-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

V.2 Conjunto de datos sintéticos

Para seleccionar los mejores kernels en cada uno de los experimentos con datos sintéticos, se procedió a evaluar dos aspectos: la tendencia de crecimiento con respecto al parámetro a evaluar (*densidad*, *número* y *tamaño*) y el tiempo de ejecución. De esta manera, primeramente se calculó la razón de cambio del tiempo de ejecución con respecto a dicho parámetro de cada uno de los kernels y se ordenó de manera ascendente.

Se seleccionó el de menor *pendiente* o *razón de cambio* de cada uno de los dos tipos principales (*caminos más cortos* y *subárboles*) y solo se tomó en cuenta el tiempo de ejecución como criterio de desempate en caso de que se tuvieran varios con la misma pendiente.

Se le da prioridad a la *razón de cambio* pues se considera que la estabilidad en tiempo de ejecución con respecto a la *densidad* de los grafos, el *número* de grafos y el *tamaño* de los mismos es más importante por el hecho de que es más común trabajar con conjuntos de datos heterogéneos en estos aspectos.

A continuación se describen los resultados para los tres experimentos realizados con conjuntos de datos sintéticos.

V.2.1 Mejores resultados con respecto a la *densidad* de los grafos

Para el experimento en el cual la *densidad* de los grafos de entrada es la variable principal, los métodos kernel que obtuvieron *mejores* resultados son: el kernel de *conteo de caminos más cortos podado* en su variante *polinomial* (*SPC-T-poli*) y el kernel de *subárboles* sobre grafos de caminos más cortos en su variante *RBF* (*WL-FW-rbf*).

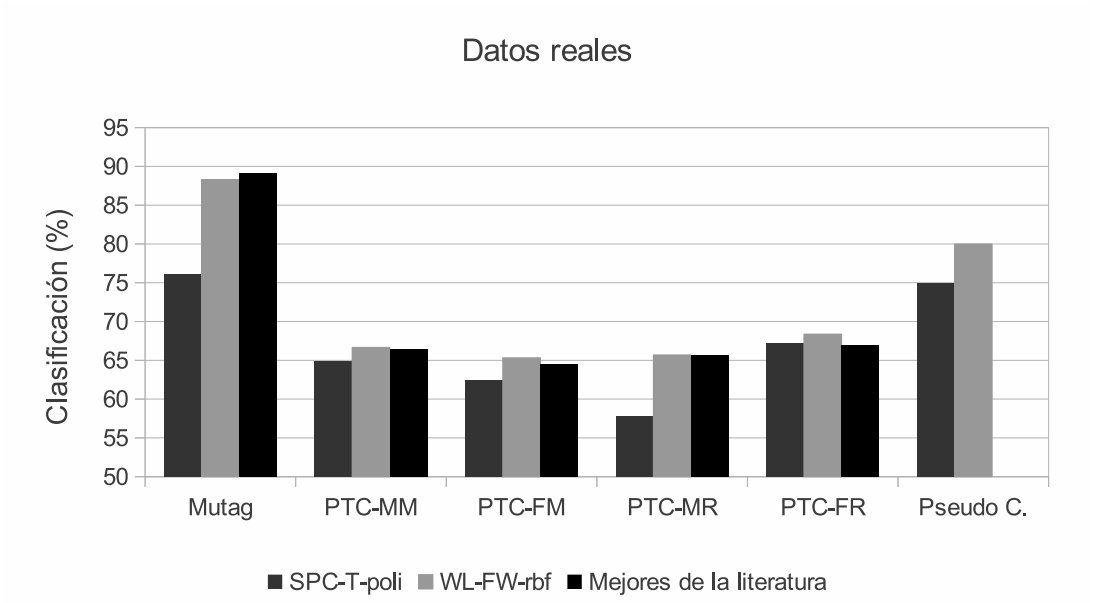
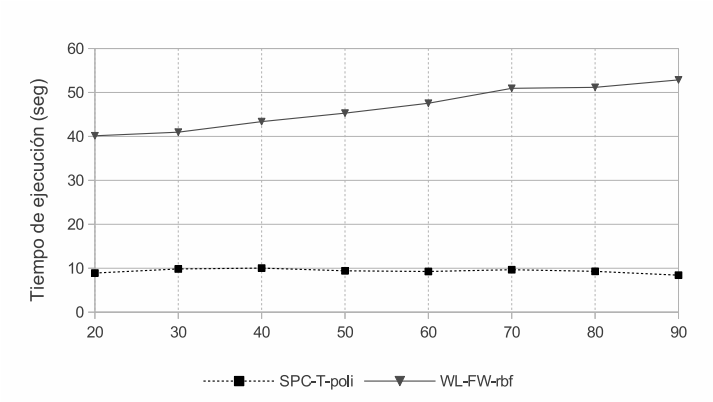


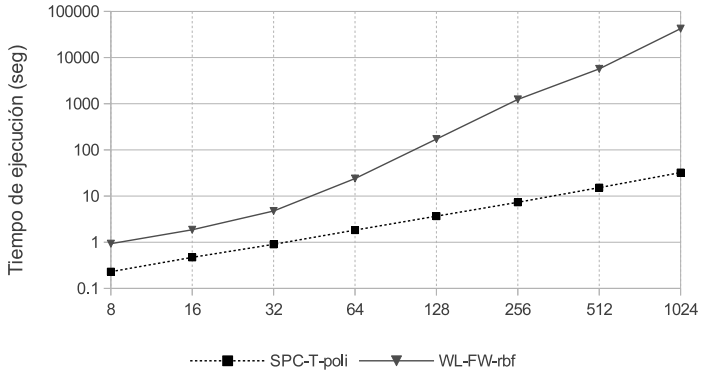
Figura 35. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernels con mejor desempeño con respecto a la densidad: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*), subárboles sobre grafos de caminos más cortos variante RBF (*WL-FW-rbf*), kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.

V.2.1.1 Kernel de caminos más cortos *SP* vs kernel de subárboles *WL*

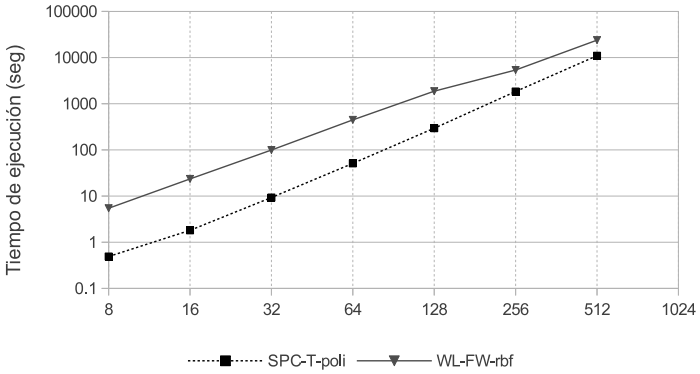
En la Figura 35 se puede ver cómo el kernel *WL-FW-rbf* superó en todos los casos al kernel *SPC-T-poli* y en 3 de los 5 casos por más de 5%. La diferencia máxima entre los porcentajes de clasificación de ambos es de 12.2%.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 36. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos (**N** = 64) y nodos por grafo (**n** = 32), (b) el número de grafos **N**, para un número de nodos por grafo fijo (**n** = 16) y densidad de aristas (**c** = 50%) y (c) tamaño de grafos **n**, para un número fijo de grafos (**N** = 64) y densidad de aristas (**c** = 50%). Kernels con mejor desempeño con respecto a la densidad: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles sobre grafos de caminos más cortos variante RBF (*WL-FW-rbf*).

Los resultados de los experimentos con conjuntos de datos sintéticos son favorables a *SPC-T-poli* ya que en todos los casos (Figura 36), además de tener un menor tiempo de ejecución, la tendencia de crecimiento con respecto a las distintas variables es menos pronunciada que en el caso de *WL-FW-rbf*. Esto es: 0 vs 0.2, 0.03 vs 38.77 y 20.84 vs 45.88 para las pendientes promedio de *SPC-T-poli* y *WL-FW-rbf* en las Figuras 36(a), 36(b) y 36(c), respectivamente.

Por último, en el caso del experimento de clasificación por *esquemas de etiquetado* (Figura 37), el kernel *SPC-T-poli* resulta *superior* ya que obtiene porcentajes de clasificación mayores que los de *WL-FW-rbf* en todos los casos con excepción de dos (de los cuales solo uno por una diferencia importante). Incluso en algunos casos, la diferencia a favor del kernel *SPC-T-poli* es superior a los 20 puntos porcentuales llegando en su punto máximo a 38% a favor, lo que lo hace superior a *WL-FW-rbf*.

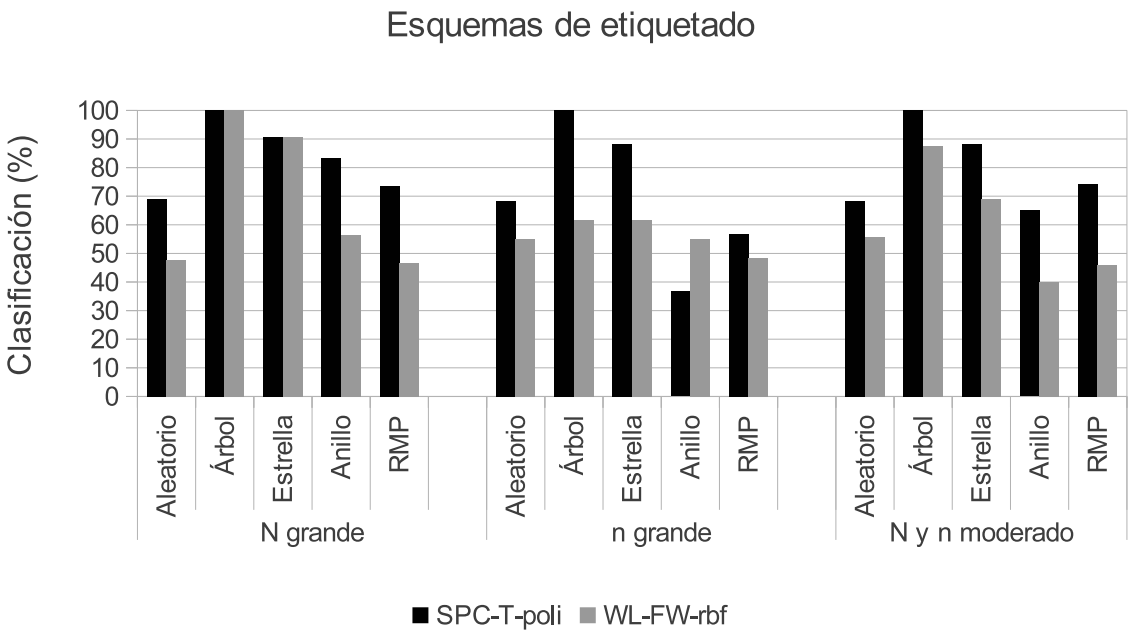


Figura 37. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernels con mejor desempeño con respecto a la densidad: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles sobre grafos de caminos más cortos variante RBF (*WL-FW-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

V.2.2 Mejores resultados con respecto a la *número* de grafos

Para el experimento en el cual el *número* de grafos de entrada (N) es la variable principal, los métodos kernel que obtuvieron *mejores* resultados son: el kernel de conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y el kernel de subárboles sobre grafos de caminos más cortos considerando aristas, variante RBF (*WLE-FW-rbf*).

V.2.2.1 Kernel de caminos más cortos *SP* vs kernel de subárboles *WL*

Al comparar los kernels de los dos tipos principales (*caminos más cortos* y *subárboles*) nos podemos dar cuenta que sucede casi lo mismo que con los kernels con mejor desempeño con respecto a la *densidad*.

Por un lado, para el experimento de los conjuntos de datos reales (Figura 38) el kernel de subárboles *WLE-FW-rbf* supera al kernel de *caminos más cortos SPC-T-poli* en todos los conjuntos de la misma manera que en la Figura 35.

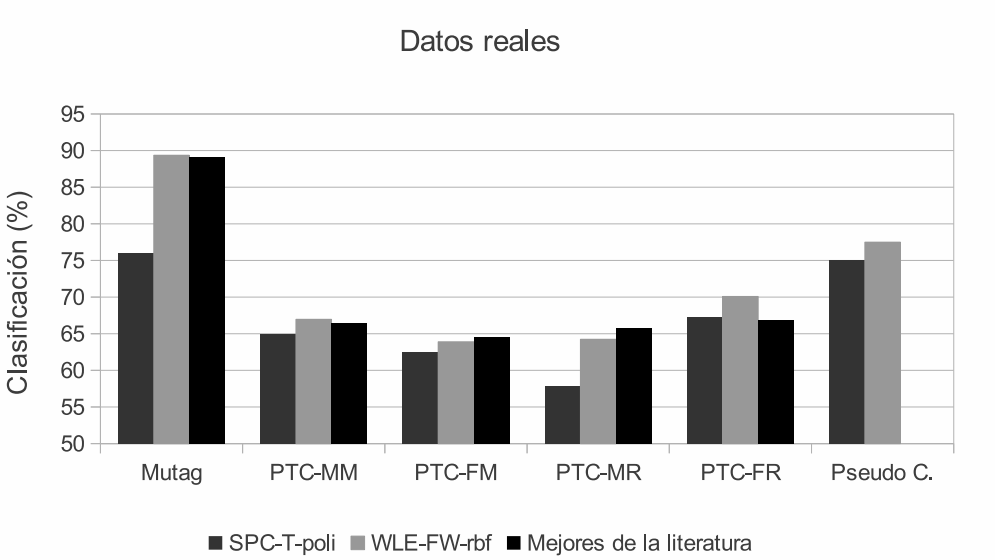


Figura 38. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernels con mejor desempeño con respecto al número de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*), subárboles sobre grafos de caminos más cortos considerando aristas variante RBF (*WLE-FW-rbf*), kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.

Para los experimentos con conjuntos de datos sintéticos (Figura 39), el resultado es el mismo. El kernel de *caminos más cortos SPC-T-poli* supera al kernel *WLE-FW-rbf* tanto en tiempo de ejecución como en la tendencia de cambio con respecto a las distintas variables (*densidad*, *número* y *tamaño*) pues crece de manera menos pronunciada. Esto es: 0 vs 0.33, 0.03 vs 11.69 y 20.84 vs 37.69 para las pendientes promedio de *SPC-T-poli* y *WLE-FW-rbf* en las Figuras 39(a), 39(b) y 39(c), respectivamente.

Por último, para el experimento en el que se clasifica por *esquemas de etiquetado* (Figura 40), se tiene un comportamiento similar, en el que el kernel de *caminos más cortos SPC-T-poli* obtiene mejores resultados que el kernel de *subárboles WL-FW-rbf*. Sin embargo, la superioridad del kernel de *caminos más cortos* no es tan evidente como en la Figura 37 pues el número de casos en los cuales se tiene una diferencia importante es menor y además, en promedio esta diferencia también es menor.

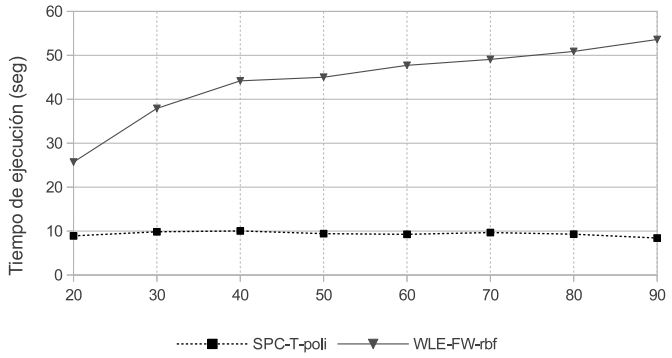
V.2.3 Mejores resultados con respecto al *tamaño* de los grafos

Para el experimento en el cual se tomó como variable el *tamaño* de los grafos de entrada, los kernels con mejor desempeño son: el kernel de conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y el kernel de subárboles considerando aristas, variante RBF (*WLE-rbf*).

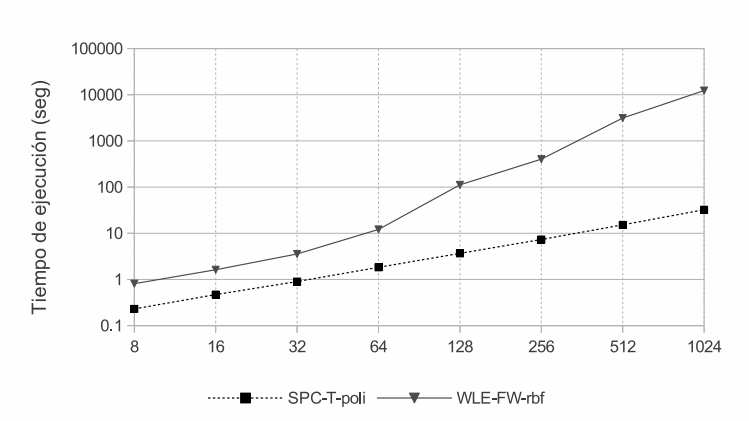
V.2.3.1 Kernel de caminos más cortos *SP* vs kernel de subárboles *WL*

En la Figura 41 se muestra la comparación entre los kernels *SPC-T-poli* y *WLE-rbf* aplicados a la clasificación de los conjuntos reales. Se puede ver cómo el segundo supera en todos los casos al primero. En promedio, la diferencia entre los porcentajes de clasificación de ambos es de 5% mientras que la diferencia máxima es de 11.1%.

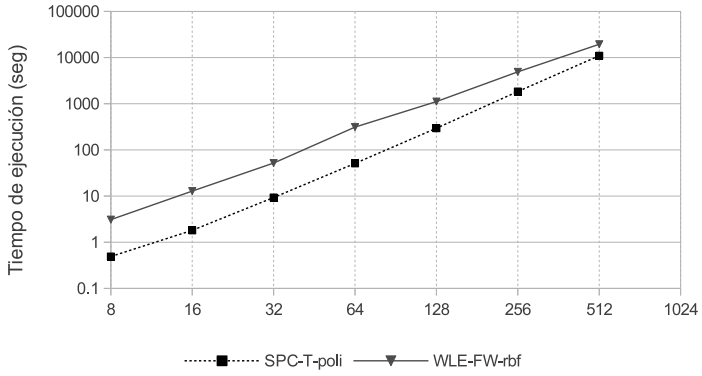
Para los resultados de los experimentos con conjuntos de datos sintéticos (Figura 42), se tiene el mismo comportamiento que los observados en los experimentos de los otros dos parámetros (*densidad* y *número*) excepto con el experimento en donde se varía el *tamaño* de los grafos (Figura 42(c)) ya que, aunque los valores de la mayoría de los casos son menores



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 39. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernels con mejor desempeño con respecto al número de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles sobre grafos de caminos más cortos considerando aristas variante RBF (*WLE-FW-rbf*).

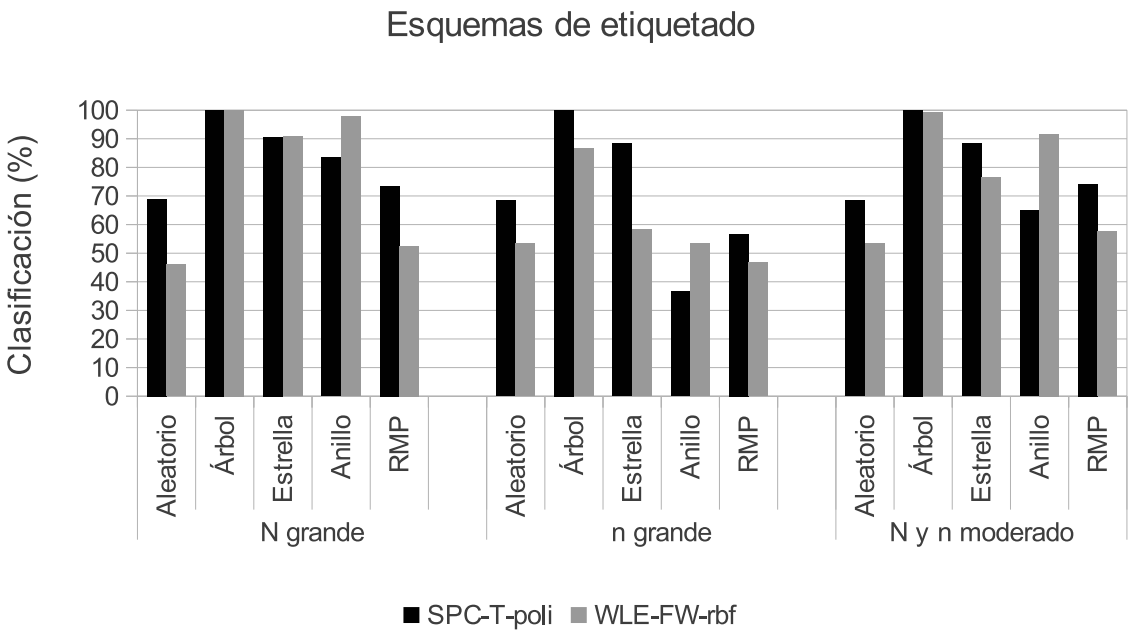


Figura 40. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernels con mejor desempeño con respecto al número de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles sobre grafos de caminos más cortos considerando aristas variante RBF (*WLE-FW-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

para *SPC-T-poli*, la tendencia indica que *WLE-rbf* resulta mejor para más grafos ya que su razón de cambio es menor (pendientes promedio de 20.84 para *SPC-T-poli* vs 17.16 para *WLE-rbf*). De hecho, no fue posible calcular el caso con 1024 grafos para el kernel *SPC-T-poli* mientras que si fue posible con *WLE-rbf*.

Para el experimento en el que se clasifique por *esquemas de etiquetado* (Figura 43), el kernel *SPC-poli* tiene un desempeño igual o mejor que *WLE-rbf* en todos los casos menos en el caso en el que la topología de los grafos es tipo *anillo* en donde este es superado. En ese caso, el kernel *WLE-rbf* llega a ser hasta 43% superior que *SPC-T-poli*.

V.3 Esquemas de etiquetado

En el caso del experimento en el cual se evalúa el desempeño de clasificación por esquemas de etiquetado, los kernels que tuvieron un mejor desempeño son: el kernel de conteo de caminos más cortos variante polinomial (*SPC-poli*) y el kernel subárboles sobre grafos de

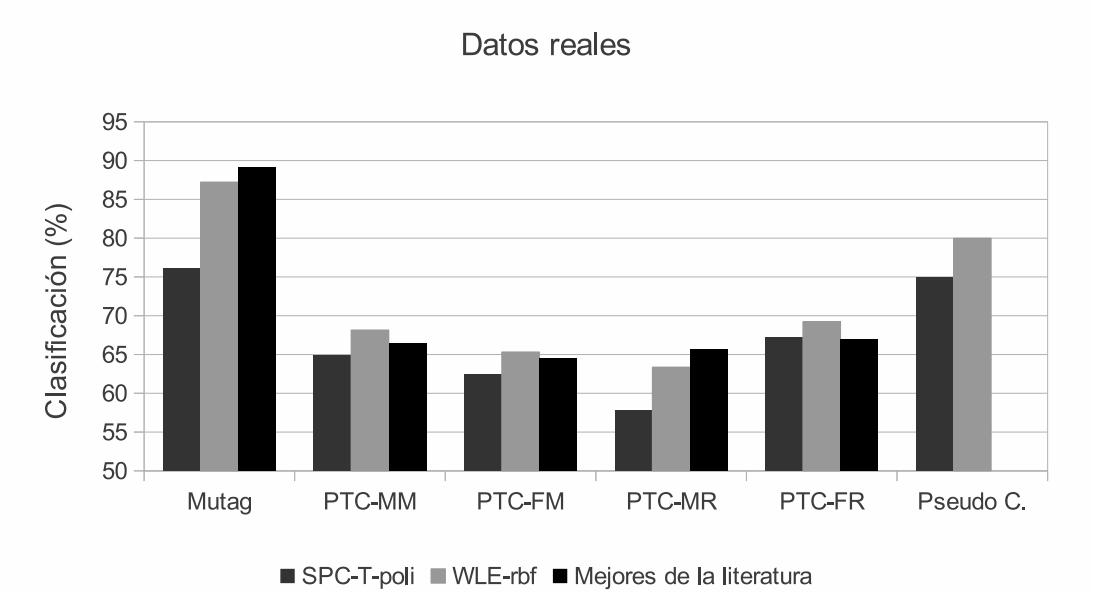


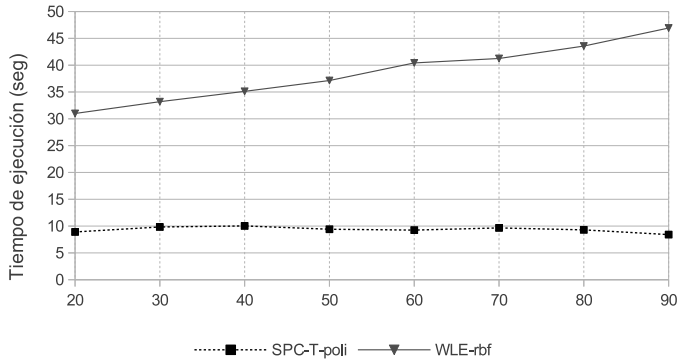
Figura 41. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernels con mejor desempeño con respecto al tamaño de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*), subárboles considerando aristas variante RBF (*WLE-rbf*), kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.

caminos más cortos *podados* tomando en cuenta aristas, en su variante lineal (*WLE-FW-T-lineal*).

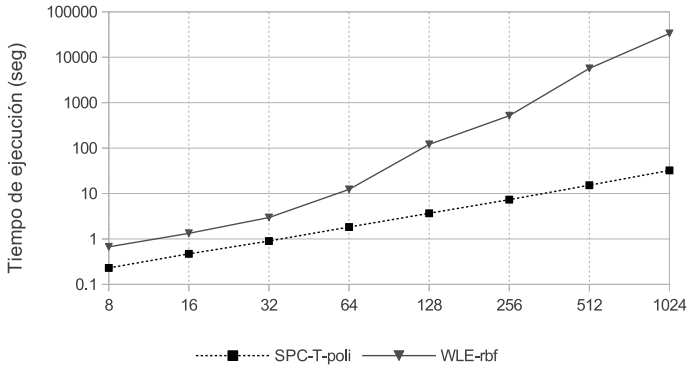
Estos kernels se seleccionaron siguiendo el mismo criterio que para el experimento que se realizó con los conjuntos de datos reales, es decir, se promedió la totalidad de los porcentajes de clasificación para cada método kernel y se tomó el de mayor promedio para cada uno de los dos tipos principales (*caminos más cortos* y *subárboles*).

V.3.0.2 Kernel de caminos más cortos *SP* vs kernel de subárboles *WL*

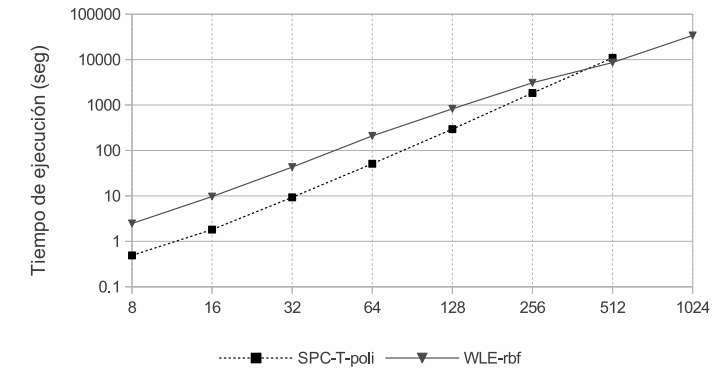
En la Figura 44 se puede observar como en este caso no resulta fácil determinar a un kernel como *mejor* entre *SPC-poli* y *WLE-FW-T-lineal*. Por un lado, en promedio *SPC-poli* es mejor que *WLE-FW-T-lineal* además de ser superior por un valor máximo de 7.5% en una de las instancias del experimento. Sin embargo, en otros casos, el segundo es superior a *SPC-poli* aunque por un máximo de 1.7%.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 42. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernels con mejor desempeño con respecto al tamaño de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles considerando aristas variante RBF (*WLE-rbf*).

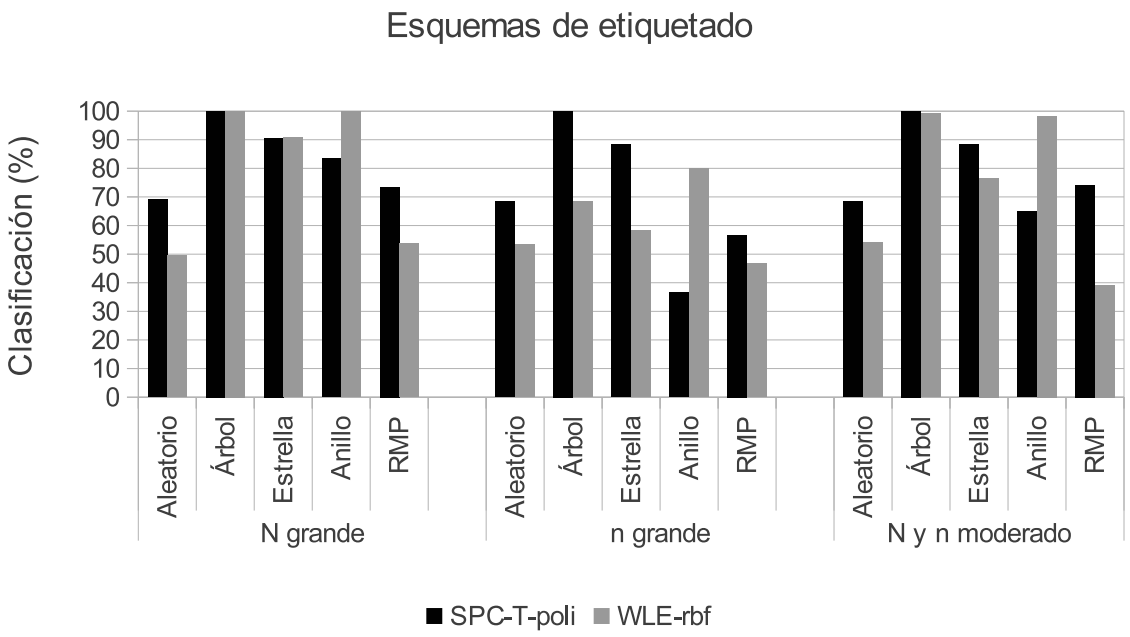


Figura 43. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernels con mejor desempeño con respecto al tamaño de grafos: conteo de caminos más cortos *podado* variante polinomial (*SPC-T-poli*) y subárboles considerando aristas variante RBF (*WLE-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

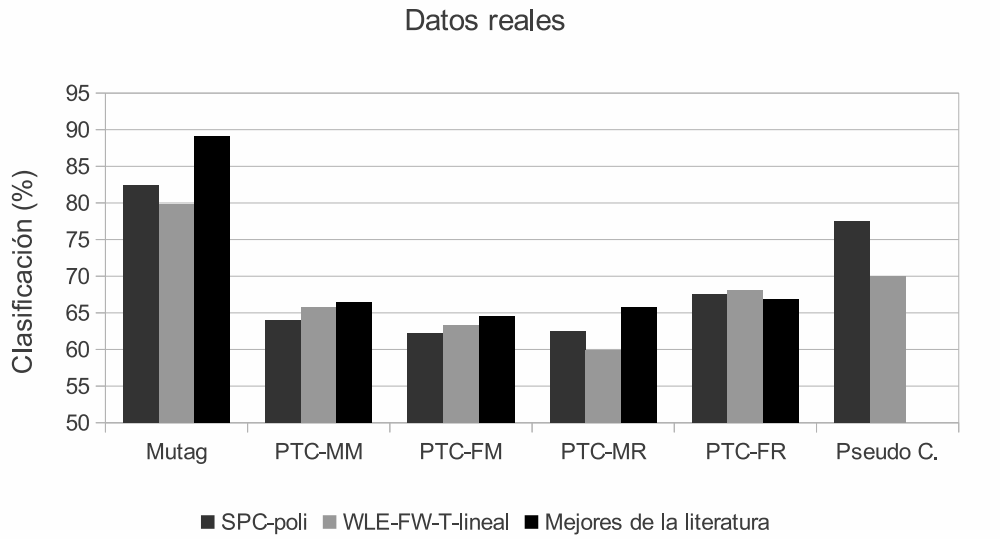


Figura 44. Porcentaje de clasificación en *Mutag*, *PTC* y *Pseudocentros*. Kernels con mejor desempeño para esquemas de etiquetado: conteo de caminos más cortos variante polinomial (*SPC-poli*), subárboles sobre grafos de caminos más cortos *podados* tomando en cuenta aristas variante lineal (*WLE-FW-T-lineal*), kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.).

Para el experimento en el que se clasifica por *esquemas de etiquetado* (Figura 45), se puede ver que en todos los casos a excepción de cuando la topología de los grafos es tipo *anillo*, el kernel *SPC-poli* tuvo mejores resultados con diferencias tan grandes como 41%. En el único caso en el que *WLE-FW-T-lineal* es superior, la mayor diferencia es de 5.8%.

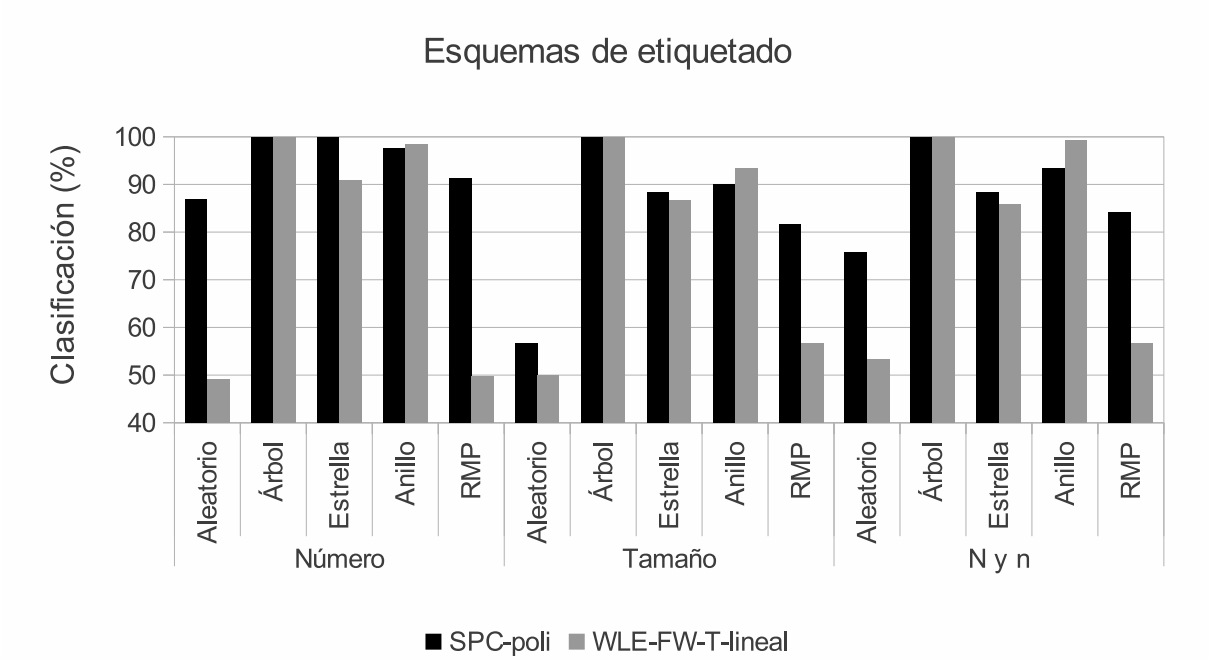
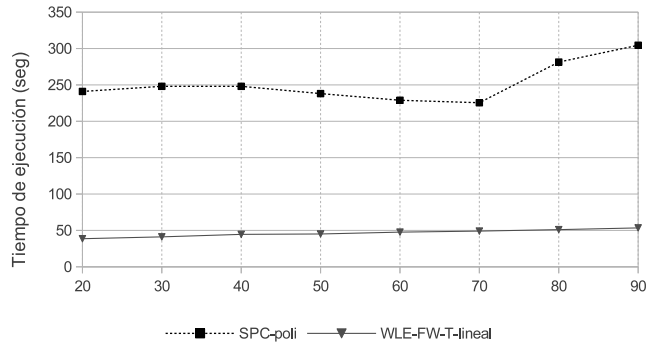


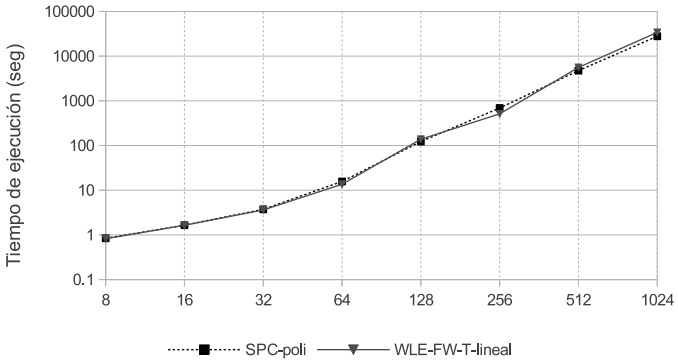
Figura 45. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernels con mejor desempeño para esquemas de etiquetado: conteo de caminos más cortos variante polinomial (*SPC-poli*) y subárboles sobre grafos de caminos más cortos *podados* tomando en cuenta aristas variante lineal (*WLE-FW-T-lineal*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

En la Figura 46 se muestra el mismo comportamiento que el de la Figura 33 en donde se comparan los mejores kernels en la clasificación de datos reales. Tanto para los experimentos de la Figura 45 como los de la Figura 46(c), el kernel *WLE-FW-T-lineal* es superior a *SPC-poli* tanto en tiempo de ejecución como en la propia tendencia de crecimiento de estos tiempos con respecto a sus respectivas variables.

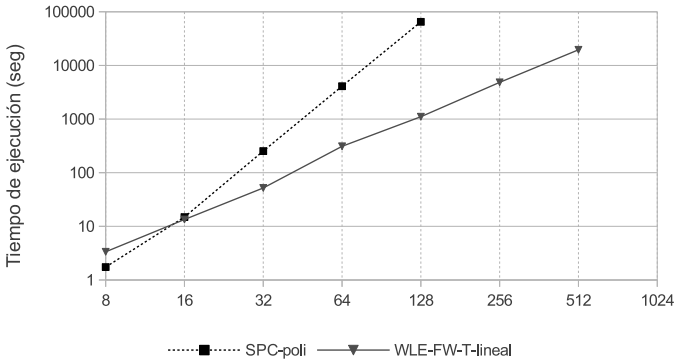
Solamente en el caso de la Figura 46(b), en donde se evalúa el tiempo de ejecución con respecto al número de grafos es en donde los resultados son básicamente iguales.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 46. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernels con mejor desempeño para esquemas de etiquetado: conteo de caminos más cortos variante polinomial (*SPC-poli*), subárboles sobre grafos de caminos más cortos *podados* tomando en cuenta aristas variante lineal (*WLE-FW-T-lineal*), kernel de descubrimiento de patrones de Kramer y Readt (2001) para Mutag y los kernels Tanimoto y MinMax de Swamidass *et al.* (2005) para PTC.

Capítulo VI

Conclusiones

VI.1 Resumen

En este trabajo se estudiaron y evaluaron distintos tipos de métodos kernel para grafos con la finalidad de realizar una comparación entre ellos y determinar cuáles son los que tienen un mejor desempeño tanto en porcentaje de clasificación como en tiempo de ejecución.

Se hizo una revisión bibliográfica de los métodos kernels en la literatura y se profundizó en tres: *caminatas aleatorias* de Vishwanathan *et al.* (2010), *caminos más cortos* de Borgwardt y Kriegel (2005) y *sub-árboles de Weisfeiler-Lehman* de Shervashidze *et al.* (2011).

Se implementaron los tres tipos de kernels pero debido a sus características además de las limitaciones implícitas de la implementación del kernel de *caminatas aleatorias*, se terminó por seleccionar los kernels de *caminos más cortos* y *sub-árboles* para su evaluación experimental. Además, se implementaron una serie de variantes a estos kernels para buscar mejorar ya sea el tiempo de ejecución o el porcentaje de clasificación de dichos kernels.

Se realizaron experimentos con dos grupos de datos distintos: datos reales y datos sintéticos. El primer grupo se compone de tres conjuntos de datos reales que se utilizan en la literatura comunmente, estos son: *Mutag* (Debnath *et al.*, 1991), *PTC* (Helma *et al.*, 2001) y *Pseudocentros* (Gramada y PE, 2006). Con estos conjuntos se evaluó el desempeño en porcentaje de clasificación de los métodos kernels implementados.

Con el primer conjunto del grupo de datos sintéticos se evaluó de manera empírica el impacto que tienen tres variables en el tiempo de ejecución de los kernels: la *densidad* de los grafos de entrada, la *cantidad* de grafos en el conjunto de datos a clasificar y el *tamaño* de estos grafos medido en número de vértices.

El segundo conjunto del grupo de datos sintéticos está compuesto de grafos que fueron

etiquetados de acuerdo a dos esquemas distintos de recorridos de árboles y están hechos con el propósito de evaluar la capacidad de los kernels de clasificarlos en función a la manera en que fueron etiquetados.

Por último se hizo un análisis de los kernels de *caminos más cortos* y *sub-árboles* con mejores resultados en los distintos experimentos.

VI.2 Conclusiones

El efecto del kernel de *caminos más cortos invertido* (*iSP*) en la implementación no trae consigo ningún beneficio notable, pero tampoco trae un impacto negativo considerable.

El kernel de sub-árboles de *Weisfeiler-Lehman* considerando aristas (*WLE*) y el kernel de sub-árboles de *Weisfeiler-Lehman* normal (*WL*) tienen aproximadamente el mismo comportamiento, dicho de otra manera, considerar aristas no impacta de manera significativa en el tiempo de ejecución ni clasificación en los primeros dos experimentos. Solo en la clasificación por esquemas de etiquetado, específicamente cuando la topología de los grafos de entrada es *anillo*, el kernel *WLE* mejora la clasificación, para el resto se mantiene igual.

Para la familia de kernels que utilizan los *caminos más cortos*, el kernel de *conteo de caminos más cortos* (*SPC*) es el que obtuvo mejores resultados. Aunque no existe un solo kernel que resultó con los mejores resultados para todos los experimentos, el kernel *SPC*, en sus variantes *rbf*, *poli* y *podado* (variante *poli*) obtuvieron mejores resultados para los datos reales, esquemas de etiquetado y datos sintéticos, respectivamente.

Para la familia de variantes al kernel de *sub-árboles*, se obtuvieron resultados más variados que en el caso del kernel de *caminos más cortos*. En este caso, no existe un solo kernel que se vea superior a los demás. Sin embargo, algo que tienen en común todos los métodos que obtuvieron mejores resultados en sus respectivos experimentos es el hecho de que siempre fue aplicando antes la transformación de *Floyd* a sus grafos de entrada.

Para el kernel de sub-árboles sobre grafos de caminos más cortos (*WL-FW*), el proceso de *podar* los grafos después de la tranformación de *Floyd* (*WL-FW-T*) probó ser innecesario

ya que no ofrece ventajas sobre los grafos sin podar, aún cuando sean grafos *densos*.

Con estos resultados, nos podemos dar cuenta de la importancia de la transformación de *Floyd* sobre los grafos a ser procesados por un método kernel. La desventaja de esto es que la tranformación en sí es costosa aunque mejora el porcentaje de clasificación.

Como se puede ver en la sección de resultados, cuando se clasifica por esquemas de etiquetado, la topología de los grafos de entrada que más favorece en el porcentaje de clasificación es *árbol* o *estrella* (que es un tipo particular de árbol). Esto quizá debido a la forma en la que se implementó el etiquetado de los grafos, esto es, basado en recorridos de árboles (*preorden* y *postorden*).

Aunque se mencione de dónde provienen los conjuntos de datos reales, es difícil encontrar documentación en donde se explique a detalle de dónde se obtuvo y cómo se manejaron los datos para poder ser procesados por los métodos kernel. Debido a esto, no es posible hacer una comparación directa con los resultados reportados en la literatura.

Debido a limitantes en tamaño y número de variaciones que se implementaron, no fue posible hacer un estudio más a fondo y que esté respaldado estadísticamente (con más de una corrida por experimento). Además de esto, por la misma razón no es posible hacer una búsqueda más exaustiva de los parámetros que mejor desempeño tienen.

Debido a las características de los algoritmos implementados no es posible clasificar conjuntos de datos con grafos de tamaño considerablemente grande (miles de nodos) ya que su tiempo de ejecución asciende rápidamente con respecto a este parámetro.

VI.3 Trabajo futuro

A partir del trabajo realizado en el presente trabajo, se presentan algunas ideas como propuestas para trabajo futuro:

Implementar optimizaciones más primitivas como la multiplicación eficiente de vectores dispersos e incluir el conteo de los caminos iniciales durante la transformación de *Floyd*.

Idear esquemas más complejos y que no dependan de recorridos de árboles. Que sean

menos dependientes de la topología y con reglas más elaboradas.

Idear esquemas de *podado* más eficientes, que no se basen en recorridos y que sean únicos (por el hecho de que pueden existir varios MSTs para un mismo grafo). El *podado* actual reduce el número de aristas a $n - 1$ (por ser un árbol).

Aprovechar la rapidez de algunos de los métodos kernel probados (aún cuando no son los mejores en clasificación) para utilizarlos de forma conjunta para la clasificación, en cierta forma un encadenamiento. Ya que varios de los kernels implementados hacen uso de la transformación de *Floyd* y la información extra obtenida en ese proceso, se puede aprovechar para correr varios métodos que utilicen esa misma información de manera conjunta.

Investigar acerca del uso de estructuras más complejas (como los *caminos más cortos*) para la construcción de métodos kernels para grafos.

VI.4 Producto de investigación

Se desarrolló un marco de trabajo para probar distintos métodos kernel y recopilar su información. Esta herramienta facilita el proceso de experimentación ya que permite:

1. La calendarización de los experimentos a realizar por medio de archivos de texto en donde se puede definir para cada uno, el kernel a utilizar, el conjunto de datos a clasificar, así como varios parámetros específicos a los kernels y al tipo de experimento.

2. Recolección y formato de los resultado de los experimentos realizados en archivos de valores separados por comas (.csv).

3. Recuperación de fallos en el suministro de energía sin perder información ya que guarda el estado de los experimentos así como las matrices de kernels ya calculadas.

4. Calendarización de experimentos y recolección de los resultados de manera remota empleando servicios de almacenamiento y sincronización de archivos por internet.

Referencias bibliográficas

Aizerman, M. A., E. Braverman, y L. I. Rozonoer 1964. "The probability problem of pattern recognition learning and the method of potential functions (Probability of pattern recognition learning by robots, using algorithm based on potential function method)". Automation and Remote Control, 25:1175–1190 p.

Alpaydin, E. 2010. "Introduction to machine learning". MIT Press, Cambridge, Massachusetts; Londres, Inglaterra, segunda edition. 11 pp.

Bernstein, D. S. 2005. "Matrix Mathematics". Princeton University Press.

Bishop, C. M. 2006. "Pattern Recognition and Machine Learning". Springer.

Borgwardt, K. M. 2007. "Graph kernels". Tesis de Doctorado, Ludwig-Maximilians-Universität, München, Alemania.

Borgwardt, K. M. y H.-P. Kriegel 2005. "Shortest-Path Kernels on Graphs". En: "IEEE International Conference on Data Mining (ICDM)". IEEE. 74–81 p.

Borgwardt, K. M., C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, y H.-P. Kriegel 2005. "Protein function prediction via graph kernels". Bioinformatics, 21 Suppl 1:i47–56 p.

Boser, B. E., I. M. Guyon, y V. Vapnik 1992. "A training algorithm for optimal margin classifiers". En: Haussler, D., editor, "ACM Workshop on Computational Learning Theory", Pittsburgh. ACM Press. 144–152 p.

Cairo, O. 2002. "Estructura de datos". McGraw Hill, México.

Chang, C.-C. y C.-J. Lin 2011. "LIBSVM: a library for support vector machines". ACM Transactions on Intelligent Systems and Technology, 2(3):27:1—-27:27 p.

Cormen, T., C. Leiserson, R. Rivest, y C. Stein 2001. "Introduction to Algorithms". The MIT Press, 2nd edition.

Cristianini, N. y J. Shawe-Taylor 2000. "An introduction to support vector machines (and other kernel-based learning methods)". Cambridge University Press. 208 pp.

Csardi, G. y Tamas Nepusz 2006. "The igraph software package for complex network research". InterJournal, Complex Sy:1695 pp.

Debnath, A. K., R. L. de Compadre, G. Debnath, A. J. Shusterman, y C. Hansch 1991. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". Journal of Medicinal Chemistry, 34(2):786–797 p.

Dijkstra, E. W. 1959. "A note on two problems in connection with graphs". Numerische Mathematik, 1:269–271 p.

Duda, R. O., P. E. Hart, y D. G. Stork 2001. "Pattern classification". Wiley, New York.

Floyd, R. W. 1962. "Algorithm 97: Shortest path". Communications of the ACM, 5(6):345—- p.

Fredman, M. L. y R. E. Tarjan 1987. "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the ACM, 34(3):596–615 p.

Gardiner, J. D., A. J. Laub, J. J. Amato, y C. B. Moler 1992. "Solution of the Sylvester matrix equation AXBˆT + CXDˆT = E". ACM Trans. Math. Softw., 18(2):223–231 p.

Gärtner, T., P. A. Flach, y S. Wrobel 2003. "On Graph Kernels: Hardness Results and Efficient Alternatives". En: Schölkopf, B. y M. K. Warmuth, editores, "Annual Conference of Computational Learning Theory". 129–143 p.

Golub, G. H. y C. F. Van Loan 1996. "Matrix computations". Johns Hopkins University Press, Baltimore.

Gramada, A. y B. PE 2006. "Multipolar representation of protein structure.". BMC bioinformatics, 7.

Hastie, T., R. Tibshirani, y J. H. Friedman 2001. "The elements of statistical learning : data mining, inference, and prediction". Springer, New York.

Haussler, D. 1999. "Convolution Kernels on Discrete Structures". Reporte técnico , Computer Science Department, University of California, Santa Cruz, California.

Helma, C., R. D. King, S. Kramer, y A. Srinivasan 2001. "The Predictive Toxicology Challenge 2000-2001". Bioinformatics, 17(1):107–108 p.

Imrich, W. y S. Klavzar 2000. "Product graphs: structure and recognition". Wiley.

Jolliffe, I. T. 1986. "Principal component analysis". Springer-Verlag, New York.

Jungnickel, D. 2008. "Graphs, networks, and algorithms". Springer, Berlin; New York.

Kashima, H., K. Tsuda, y A. Inokuchi 2003. "Marginalized Kernels Between Labeled Graphs". International Conference on Machine Learning, 20:321–328 p.

Kashima, H., K. Tsuda, y A. Inokuchi 2004. "Kernels for graphs". En: "Kernel Methods in Computational Biology". MIT Press, capítulo 7, 155–170 p.

Kondor, R. y K. Borgwardt 2008. "The Skew Spectrum of Graphs". International Conference on Machine Learning, (Conf 25):496–503 p.

Kondor, R. I. y J. Lafferty 2002. "Diffusion Kernels on Graphs and Other Discrete Input Spaces". En: "International Conference on Machine Learning".

Kramer, S. y L. D. Readt 2001. “Feature construction with version spaces for biochemical applications”. En: “International Conference on Machine Learning”. 258 – 265 p.

Kubinyi, H. 2003. “Drug research: myths, hype and reality.”. Nature reviews. Drug discovery, 2(8):665–668 p.

Kumar, R., J. Novak, y A. Tomkins 2006. “Structure and evolution of online social networks”. En: “ACM SIGKDD international conference on Knowledge discovery and data mining”, New York, New York, USA. ACM Press. 611 pp.

Leslie, C., E. Eskin, y W. S. Noble 2002. “The spectrum kernel: a string kernel for SVM protein classification.”. En: “Pacific Symposium on Biocomputing.”. 564–75 p.

Lewis, D., T. Jebara, y W. S. Noble 2006. “Support vector machine learning from heterogeneous data: an empirical analysis using protein sequence and structure.”. Bioinformatics, 22(22):2753 – 2760 p.

Mahé, P., N. Ueda, T. Akutsu, J.-L. Perret, y J.-P. Vert 2004. “Extensions of Marginalized Graph Kernels”. En: “International Conference on Machine Learning”, New York, New York, USA. ACM Press. 70 pp.

Noble, W. S. 2006. “What is a support vector machine?”. Nature biotechnology, 24(12):1565–1567 p.

Nocedal, J. y S. J. Wright 1999. “Numerical optimization”. Springer, New York.

Pegg, E. J. 2011. “Small World Network”. Mathworld - A Wolfram Web Resource.

Ramon, J. y T. Gärtner 2003. “Expressivity versus Efficiency of Graph Kernels”. En: “First International Workshop on Mining Graphs, Trees and Sequences”. 65–74 p.

Schölkopf, B. y A. J. Smola 2002. “Learning with Kernels”. MIT Press.

Sharan, R. y T. Ideker 2006. “Modeling cellular machinery through biological network comparison.”. Nature biotechnology, 24(4):427–433 p.

Shawe-Taylor, J. y N. Cristianini 2004. “Kernel methods for pattern analysis”. Cambridge University Press, Cambridge, UK.

Shervashidze, N. y K. M. Borgwardt 2009. “Fast subtree kernels on graphs”. En: Bengio, Y., D. Schuurmans, J. Lafferty, C. K. I. Williams, y A. Culotta, editores, “Advances in Neural Information Processing Systems”. 1660–1668 p.

Shervashidze, N., P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, y K. M. Borgwardt 2011. “Weisfeiler-Lehman graph kernels”. Journal of Machine Learning Research, 12:2539–2561 p.

Shulman-Peleg, A., R. Nussinov, y H. J. Wolfson 2004. “Recognition of Functional Sites in Protein Structures”. Journal of molecular biology, 339(3):607–633 p.

Swamidass, S. J., J. Chen, J. Bruand, P. Phung, L. Ralaivola, y P. Baldi 2005. "Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity.". Bioinformatics, 21 Suppl 1(2):i359–68 p.

Tsuda, K., T. Kin, y K. Asai 2002. "Marginalized kernels for biological sequences". Bioinformatics, 18(12):268–275 p.

Van Loan, C. 2000. "The ubiquitous Kronecker product". Journal of Computational and Applied Mathematics, 123(1-2):85–100 p.

Vert, J.-P., K. Tsuda, y B. Schölkopf 2004. "A primer on kernel methods". En: "Kernel Methods in Computational Biology", number1992. MIT Press, capítulo 2, 35–70 p.

Vishwanathan, S. V. N., N. N. Schraudolph, R. I. Kondor, y K. M. Borgwardt 2010. "Graph kernels". Journal of Machine Learning Research, 11:1201–1242 p.

Washio, T. y H. Motoda 2003. "State of the art of graph-based data mining". ACM SIGKDD Explorations Newsletter, 5(1):59 pp.

Weisfeiler, B. y A. A. Lehman 1968. "A reduction of a graph to a canonical form and an algebra arising during this reduction.". Nauchno-Technicheskaya Informatsia, 9(2).

Weisstein, E. 2011a. "Graph Diameter". Mathworld - A Wolfram Web Resource.

Weisstein, E. 2011b. "Subtree". Mathworld - A Wolfram Web Resource.

Zaki, N., S. Deris, y H. Alashwal 2009. "Protein Protein Interaction Detection Based on Substring Sensitivity Measure". En: "International Journal of Biomedical Science". 148–154 p.

Apéndice A

Resultados complementarios

En esta sección se presentan los resultados de los experimentos realizados con las variantes de los kernels de caminos más cortos y subárboles que no se presentan en la sección (IV).

Estos resultados corresponden los experimentos realizados con los datos sintéticos variando la *densidad* de los grafos, el *número* de grafos y el *tamaño* de grafos independientemente así como los realizados con grafos etiquetados de acuerdo a esquemas.

A.1 Resultados de experimentos con datos sintéticos

En esta sección se presenta el resto de los resultados de los experimentos con datos sintéticos mostrados en el capítulo IV.

Primero se muestran lo resultados obtenidos con las variantes del kernel de caminos más cortos y después los resultados correspondientes a los experimentos realizados con las variantes del kernel de subárboles.

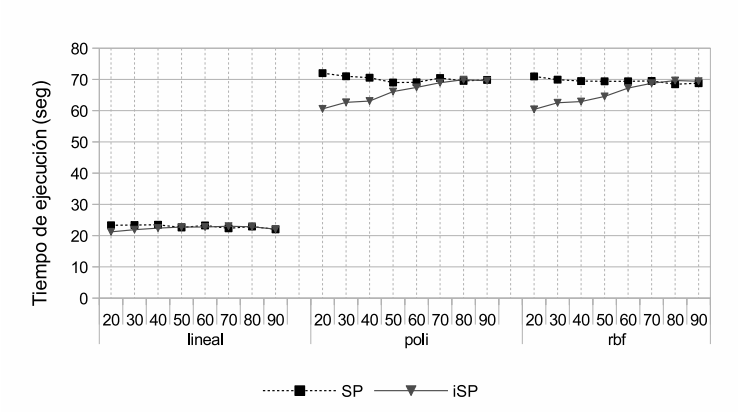
A.1.1 Variantes del kernel de caminos más cortos (SP)

A.1.1.1 Kernel de caminos más cortos invertido (iSP)

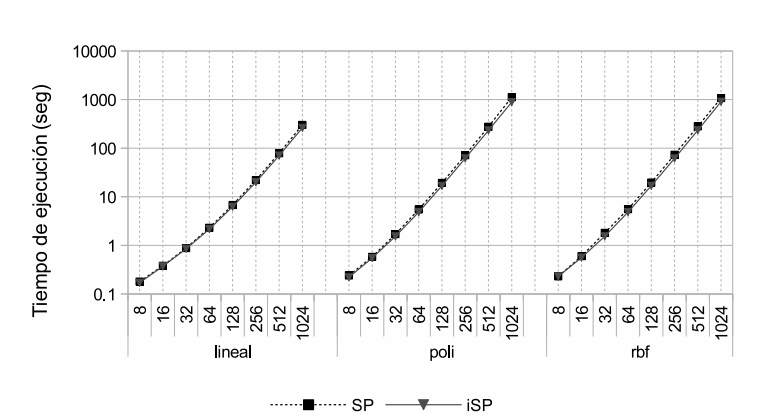
Al comparar el desempeño de tiempo de cálculo de los kernels de *caminos más cortos* (*SP*) y de *caminos más cortos invertido* (Figura 47) se puede observar la gran similitud que existe entre ambos.

Tanto para el *número* de grafos (Figura 47(b)) como para el *tamaño* de los grafos en el conjunto de datos a clasificar (Figura 47(c)), el comportamiento del tiempo de cálculo en ambos kernels (*SP* e *iSP*) en sus tres variantes (*lineal*, *poli* y *rbf*) es básicamente el mismo.

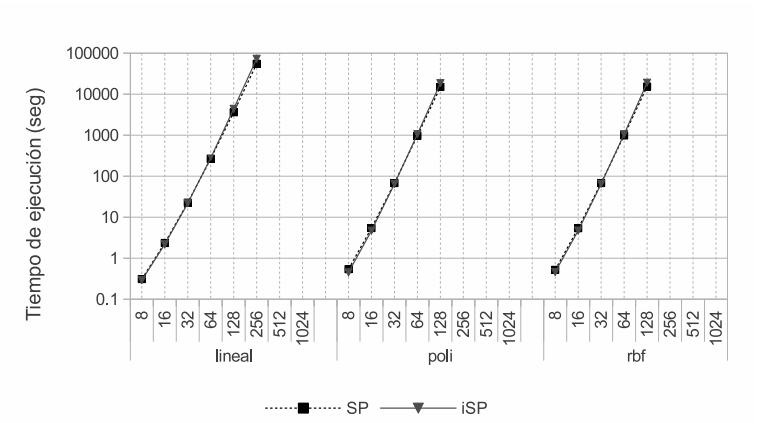
Cuando la *densidad* es lo que varía (Figura 47(a)), ambos kernels son prácticamente



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 47. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos invertido y sus variantes (*iSP-lineal*, *iSP-poli* e *iSP-rbf*).

insensibles al cambio. El kernel *iSP* en sus versiones *polinomial* y *rbf* muestran un menor tiempo de ejecución que *SP* cuando los grafos son menos densos y aumenta conforme aumenta la densidad, sin embargo, la diferencia en tiempos es pequeña y disminuye conforme la densidad aumenta hasta llegar al punto en el que se igualan.

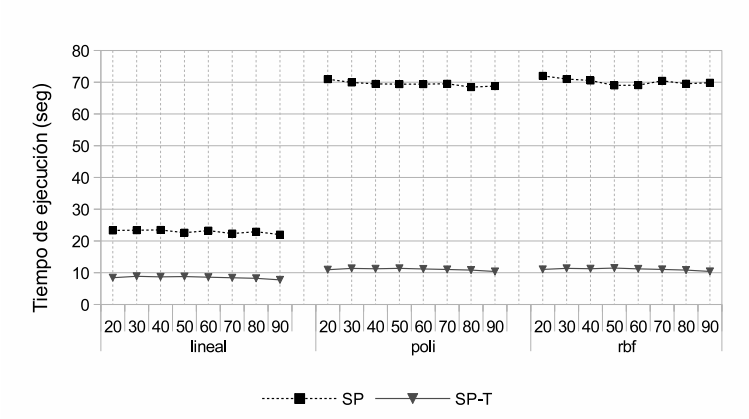
A.1.1.2 Kernel de caminos más cortos *podado* (SP-T)

Al probar el efecto de *podar* los grafos antes de aplicar el kernel de *caminos más cortos* (*SP-T*) en contra del comportamiento del kernel de *caminos más cortos* normal (*SP*), se puede ver que sí impacta considerablemente en los tiempos de cálculo con respecto a los tres parámetros a variar (*densidad*, *número* y *tamaño*).

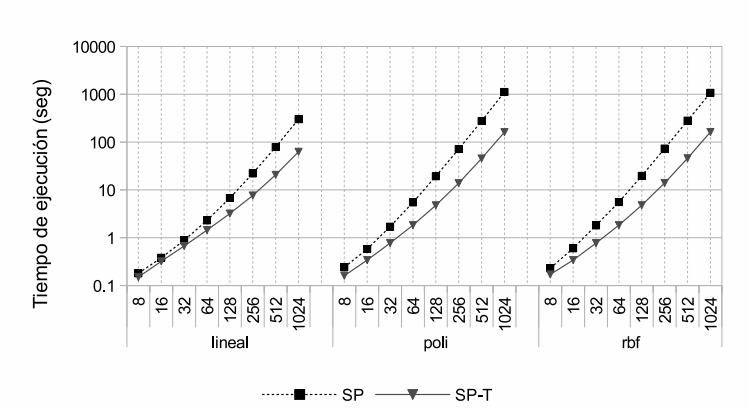
Primeramente, en la Figura 48(a) se puede ver cómo además de ser insesible al cambio de densidad como el original, el tiempo de ejecución es mucho menor que su contraparte sin *podar*. Esta diferencia se hace más evidente en las variantes *poli* y *rbf* en donde su costo de aplicación es mucho menor sobre el del kernel *lineal* a diferencia de su contraparte *SP*.

El comportamiento de ambos kernels para el caso en el que se varía el *número* de grafos en el conjunto de datos es básicamente el mismo al comenzar pero se despega rápidamente. En la Figura 48(b) se puede ver cómo la versión *podada* del kernel de *caminos más cortos* (*SP-T*) crece a un paso mucho menor que su contraparte sin podar (*SP*) en cualquiera de sus variantes.

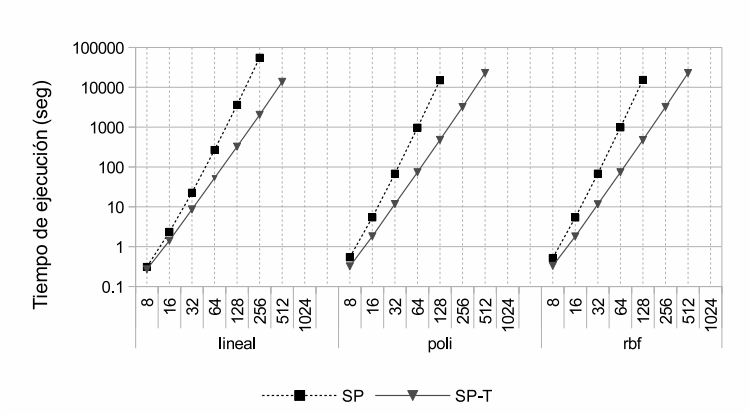
Para el caso en el que se varía el *tamaño* (Figura 48(c)) el aumento del tiempo de cálculo con respecto a ese parámetro es mucho más pronunciado que para el caso del número de grafos, tanto así que no fue posible realizar la totalidad del experimento ya que varias instancias tardaron más del límite impuesto de 48hrs. Con respecto a la diferencia en crecimiento del tiempo de cálculo entre ambos kernel y sus variantes, tenemos que también crece con más rapidez, a tal grado que fue posible procesar una instancia más del experimento (512 nodos) para el kernel *SP-T-lineal* y dos instancias más (256 y 512 nodos) para *SP-T-poli* y *SP-T-rbf* a comparación de *SP-lineal*, *SP-poli* y *SP-rbf*, respectivamente.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 48. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos *podado* y sus variantes (*SP-T-lineal*, *SP-T-poli* y *SP-T-rbf*).

A.1.1.3 Kernel de caminos más cortos invertido *podado* (iSP-T)

Los resultados de la comparación entre el kernel de *caminos más cortos* invertido (*iSP*) y su versión *podada* (*iSP-T*) resultaron ser muy parecidos a los de la versión normal del kernel de *caminos más cortos*, (*SP* y *SP-T*).

Debido a que el desempeño en tiempo de cálculo de los kernels de *caminos más cortos* y de *caminos más cortos invertido* (*SP* y *iSP* respectivamente) son similares (ver Figura 47), es esperado que el desempeño de sus variantes *podadas* (*SP-T* y *iSP-T*) tenga el mismo comportamiento (ver Figuras 48 y 49).

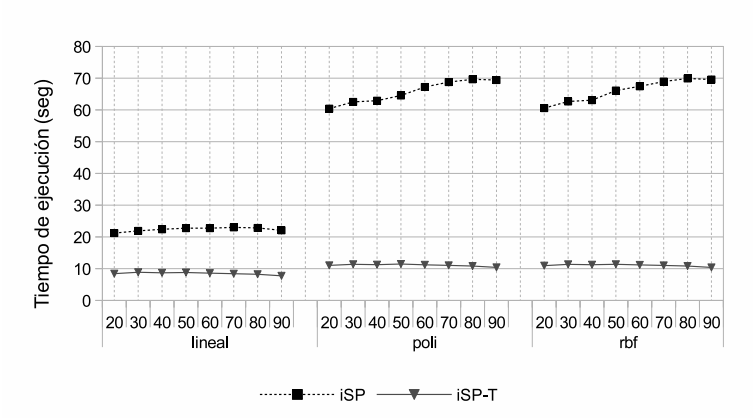
A.1.1.4 Kernel de conteo de caminos más cortos (SPC)

A diferencia de las variantes anteriores al kernel de *caminos más cortos* original (*SP*), el kernel de *conteo de caminos más cortos* (*SPC*) no ofrece mejoras sobre los tiempos de ejecución en ninguna de las pruebas (Figura 50).

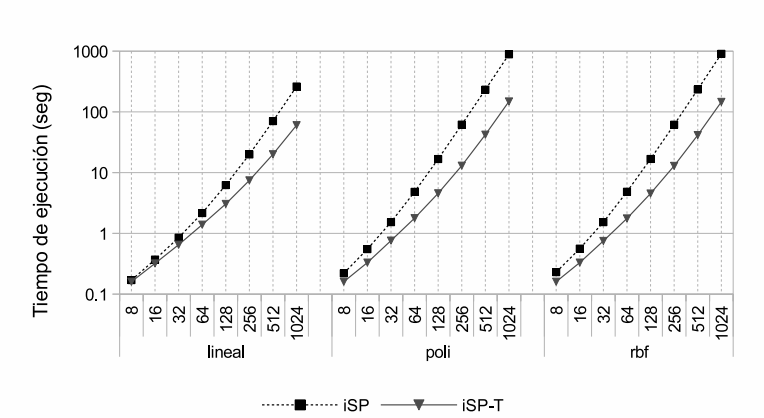
El kernel *SPC*, además de tener un tiempo de cálculo mayor al de *SP* (en cualquiera de sus variantes), este varía conforme aumenta la *densidad* de los grafos (Figura 50(a)), mientras que el original se mantiene constante.

La diferencia entre los tiempos de cálculo con respecto al *número* de grafos a procesar (Figura 50(b)) sigue con el mismo patrón, a lo largo de todas las instancias el tiempo para el kernel original (*SP*) en sus tres variantes es menor y su distancia empieza a crecer a partir del cuarto punto hasta terminar separados por un orden de magnitud en su instancia final.

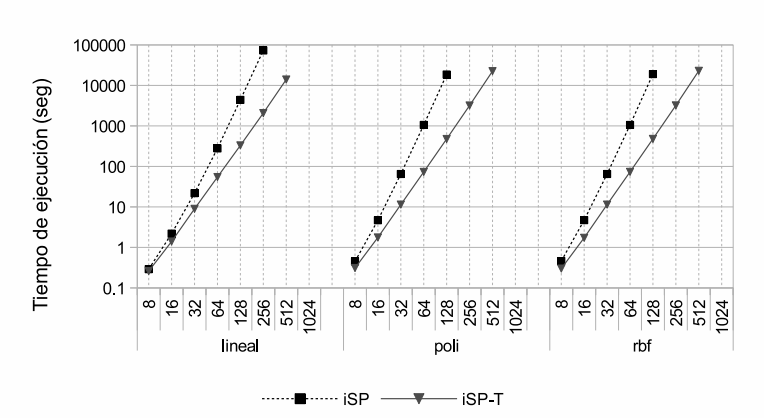
Por último, para la comparación de tiempo de cálculo con respecto al *tamaño* de los grafos entre los dos kernels *SP* y *SPC* (Figura 50(c)), se reduce la diferencia a comparación del experimento de la Figura 50(b). Se puede ver cómo los resultados guardan una distancia, siempre a favor del kernel *SP* y conforme aumenta el *tamaño* de los grafos aumenta el tiempo de cálculo con la misma proporción.



(a) Densidad de grafos **c**.

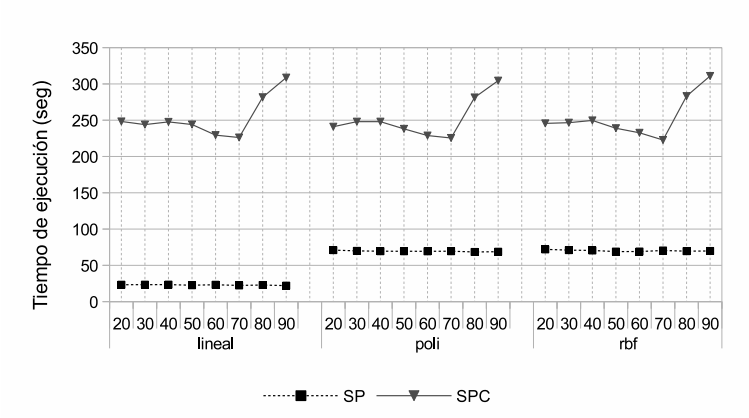


(b) Número de grafos **N**.

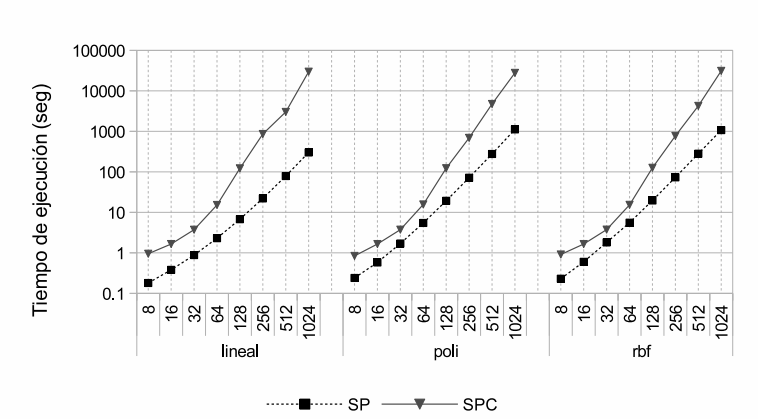


(c) Tamaño de grafos **n**.

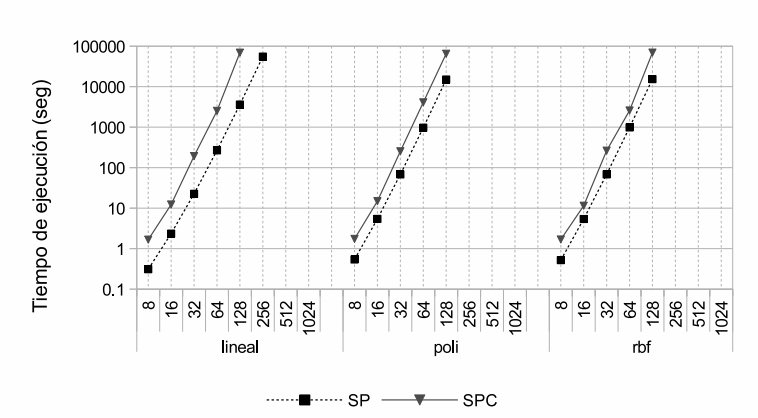
Figura 49. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de caminos más cortos invertido y sus variantes (*iSP-lineal*, *iSP-poli* e *iSP-rbf*) y kernel de caminos más cortos invertido *podado* y sus variantes(*iSP-T-lineal*, *iSP-T-poli* e *iSP-T-rbf*).



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 50. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de caminos más cortos y sus variantes (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de conteo de caminos más cortos y sus variantes(*SPC-lineal*, *SPC-poli* e *SPC-rbf*).

A.1.1.5 Kernel de conteo de caminos más cortos *podado* (SPC-T)

El efecto negativo en el desempeño de tiempo de cálculo del kernel de *conteo de caminos más cortos* (*SPC*) sobre el original (*SP*) se neutraliza al *podar* los grafos de entrada.

Se puede observar en la Figura 51(a) que el tiempo de cálculo del kernel *SPC-T* con respecto a la *densidad* de los grafos de entrada disminuye tanto en el tiempo de ejecución como en la tasa de cambio haciéndola insensible al cambio al igual que el kernel original *SP*.

El tiempo de cálculo de los kernels con respecto al *número* de grafos también mejoró al *podar* los grafos de entrada. En la Figura 51(b) se puede ver como el tiempo de cálculo aumenta con una tasa de cambio constante para el kernel *SPC-T* mientras que para el kernel *SPC* sufre un aumento significativo después de la instancia en la que se procesan 32 grafos.

Para la Figura 51(c) se tiene la misma situación en donde el tiempo de ejecución del kernel *SPC-T* es menor a lo largo de todas las instancias además de que crece a un paso mucho menos pronunciado que su contraparte sin podar (*SPC*).

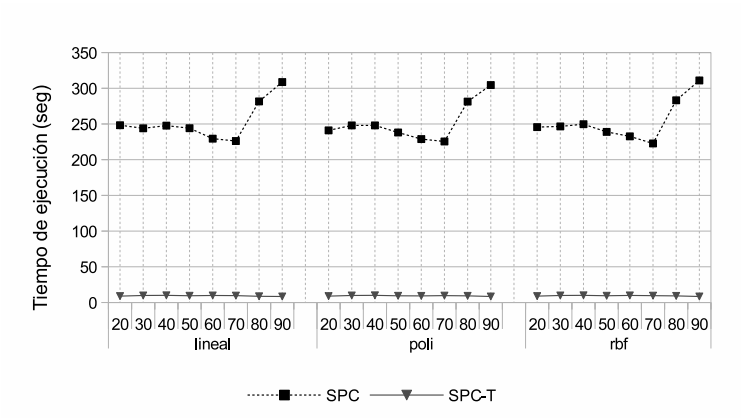
Con esto en mente, la versión podada del kernel de *conteo de caminos más cortos* ofrece una mejora en lo que respecta a tiempo de cálculo.

A.1.2 Variantes del kernel de subárboles (WL)

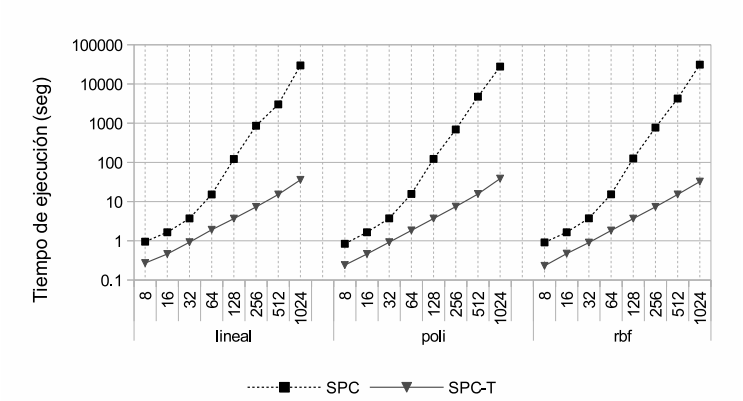
A.1.2.1 Kernel de subárboles considerando aristas (WLE)

Al igual que el experimento anterior, las diferencias de tiempo de cálculo entre el kernel de *subárboles* (*WL*) y su versión que toma en cuenta aristas (*WLE*) son despreciables.

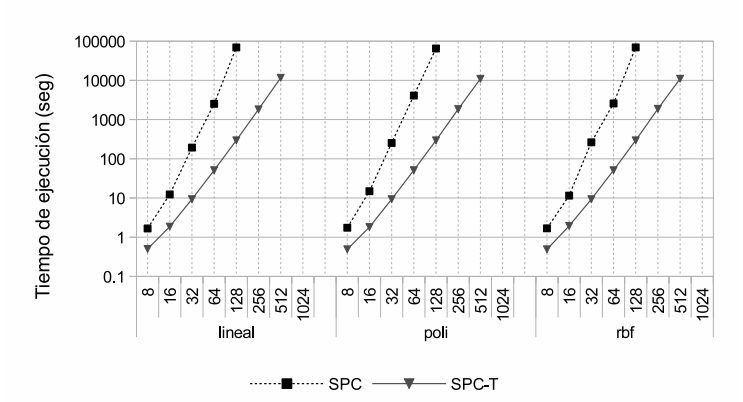
Para los experimentos de *densidad*, *número* y *tamaño* de grafos (Figuras 52(a), 52(b) y 52(c), respectivamente) el comportamiento del tiempo de cálculo es casi idéntico. Esto indica que el hecho de tomar en cuenta las aristas en el kernel no impacta en tiempos de ninguna manera, lo que significa que, en caso de que esta variación ofrezca mejores resultados de clasificación que su contraparte *WL*, se puede usar sin ningún efecto negativo



(a) Densidad de grafos **c**.

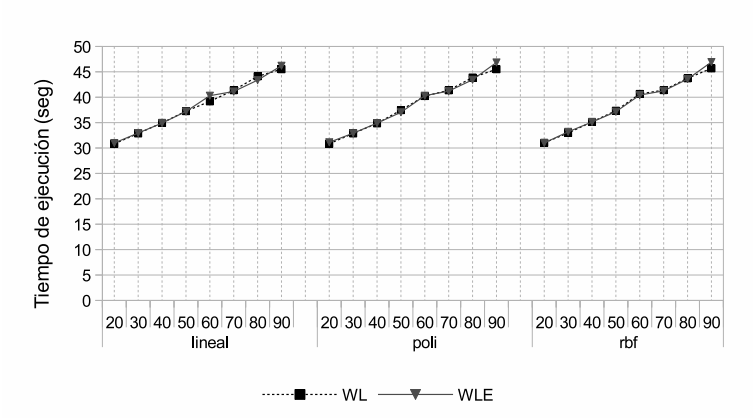


(b) Número de grafos **N**.

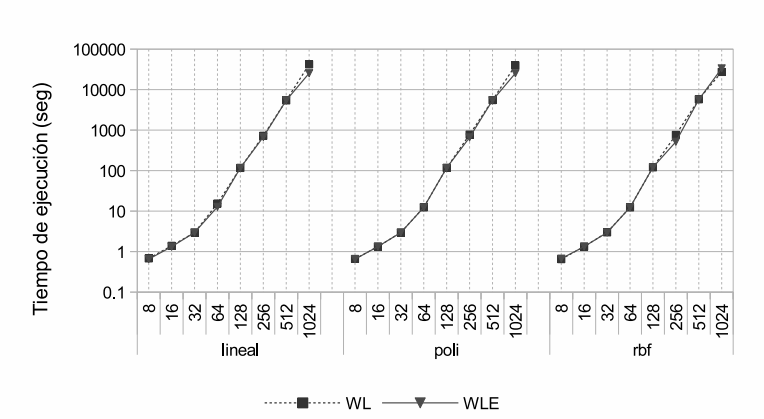


(c) Tamaño de grafos **n**.

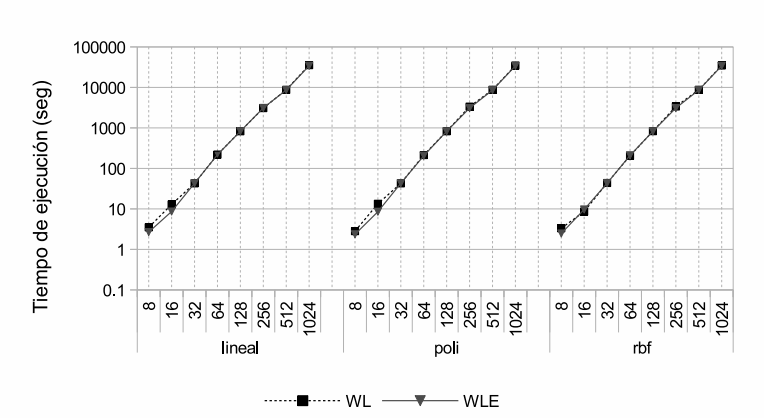
Figura 51. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de *conteo de caminos más cortos* y sus variantes (*SPC-lineal*, *SPC-poli* y *SPC-rbf*) y kernel de *conteo de caminos más cortos podado* y sus variantes(*SPC-T-lineal*, *SPC-T-poli* y *SPC-T-rbf*).



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 52. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de sub-árboles y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de sub-árboles considerando aristas y sus variantes (*WLE-lineal*, *WLE-poli* y *WLE-rbf*).

en el tiempo de ejecución.

A.1.2.2 Kernel de subárboles sobre grafos de caminos más cortos (WL-FW)

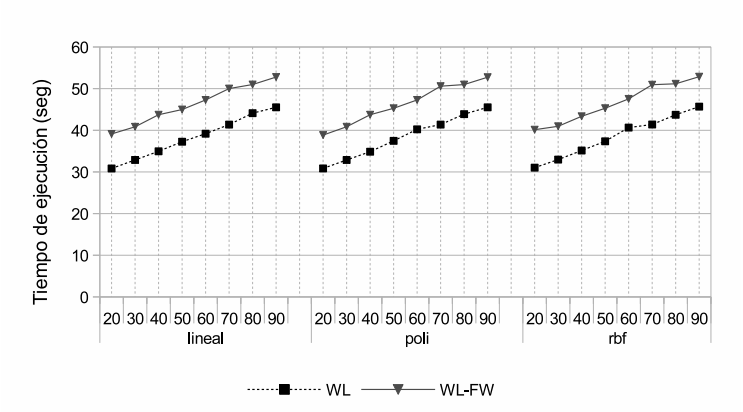
Cuando se aplica la transformación de *Floyd-Warshall* a los grafos a clasificar antes de aplicar el kernel de *subárboles* (*WL-FW*) se espera perder un poco en tiempo de ejecución buscando obtener mejor desempeño de clasificación.

En la Figura 53(a) se muestra cómo efectivamente el tiempo de cálculo del kernel original (*WL*) es menor que la versión en la que se aplica la transformación a los grafos antes de ser procesados (*WL-FW*) y además tiene el mismo comportamiento, es decir, siempre mantiene, aproximadamente la misma distancia conforme cambia la *densidad* de los grafos.

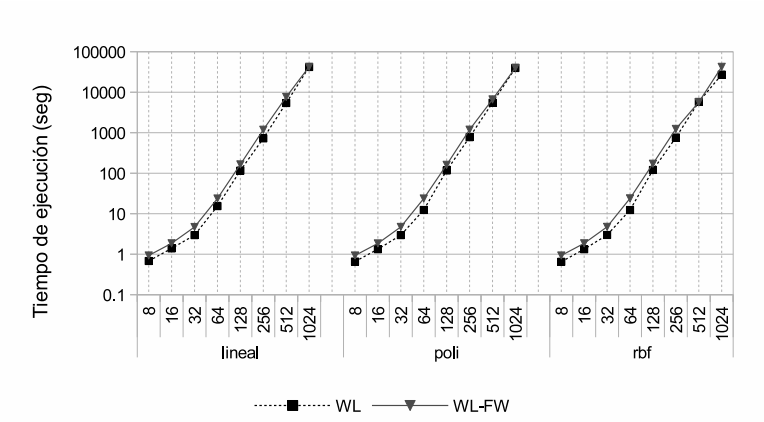
Para la Figura 53(b), en donde se varía el *número* de los grafos de entrada, se puede ver que la distancia es mucho menor entre los dos kernels y que siguen un mismo patrón, siendo la versión original (*WL*) un poco más rápida que su variación (*WL-FW*) al principio y aproximándose en las últimas casos.

El único caso del experimento en el que sí se ve un efecto negativo importante es cuando se varía el *tamaño* de los grafos de entrada (Figura 53(c)). En este caso, aunque los dos mantienen un mismo patrón de crecimiento, la diferencia fue suficiente para no poder calcular el último caso del experimento (con grafos de 1024 nodos de tamaño) pues superaba el límite de 48hrs mientras que sí fue posible para la versión original.

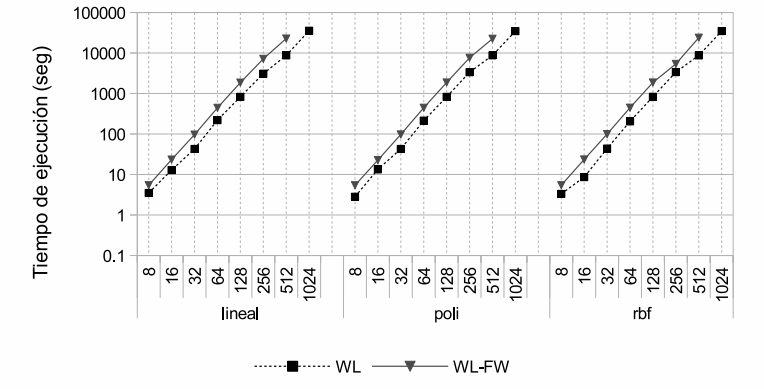
En general, la penalización que sufre el tiempo de cálculo del kernel después de aplicar la transformación de *Floyd-Warshall* no es lo suficientemente grande como para descartar la utilización de este kernel (*WL-FW*) en vez del original (*WL*), en dado caso que tuviera mejores resultados a menos que se tenga una variación importante en el tamaño de los grafos a tratar.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 53. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de *sub-árboles* y sus variantes (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variantes (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*).

A.1.2.3 Kernel de subárboles considerando aristas sobre grafos de caminos más cortos (WLE-FW)

Al comparar los kernels *WLE* y *WLE-FW* (Figura 54) se puede ver distintos comportamientos para los tres experimentos.

Primeramente, el kernel *WLE* resulta ser superior cuando la *densidad* es variable y solo en el primer caso (20% de densidad) se ve superado por su contraparte *WLE-FW* (ver Figura 54(a)).

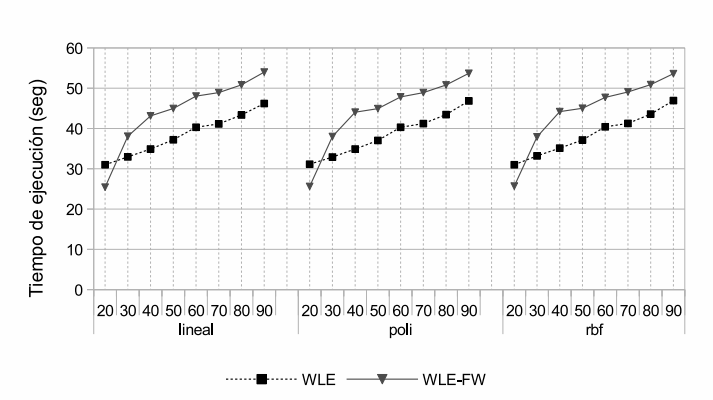
En la Figura 54(b) se aprecia que los dos métodos kernel tienen aproximadamente la misma tendencia de crecimiento, sin embargo, en un principio, el kernel *WLE-FW* tiene mejores resultados que se ven igualados y posteriormente superados lentamente por su contraparte *WLE* a partir del cuarto o quinto caso del experimento (64 o 128 grafos de entrada).

Cuando la variable es el *tamaño* de los grafos (Figura 54(c)), se tiene una tendencia a favor del kernel *WLE* ya que, aunque los resultados son muy similares entre si, se despegan lentamente a tal grado que el último caso (1024 nodos de tamaño) del experimento no fue completado por el kernel *WLE-FW* por exceder las 48hrs. mientras que *WLE* sí fue capaz de terminar.

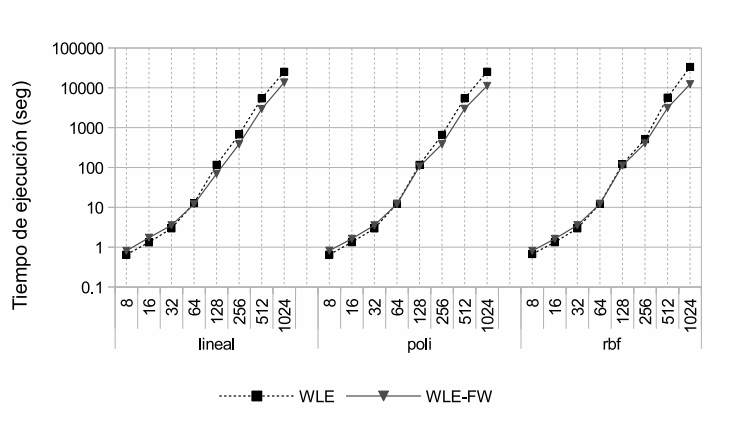
Aunque es posible distinguir qué métodos kernel son mejores para los distintos experimentos, es difícil determinar a alguno de ellos como mejor sobre el otro ya que muestran la misma tendencia de crecimiento o una que favorece a alguno de ellos en los últimos casos de dichos experimentos. Solo en caso de que se manejaran conjuntos con parámetros muy grandes (ya sea de *tamaño* o del *número* de grafos) se puede recomendar a uno sobre el otro, en caso contrario, su desempeño resulta ser el mismo.

A.1.2.4 Kernel de subárboles sobre grafos de caminos más cortos *podados* (WL-FW-T)

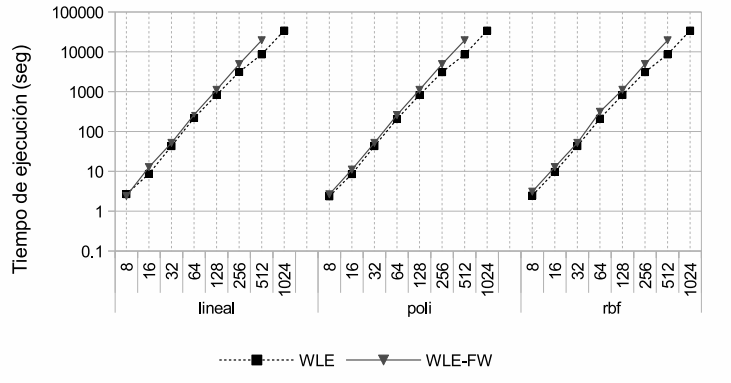
Al aplicar el proceso de *podado* a los grafos de entrada del kernel *WL-FW* (formando el kernel *WL-FW-T*), se tiene que el impacto es nulo para todos los experimentos.



(a) Densidad de grafos **c**.

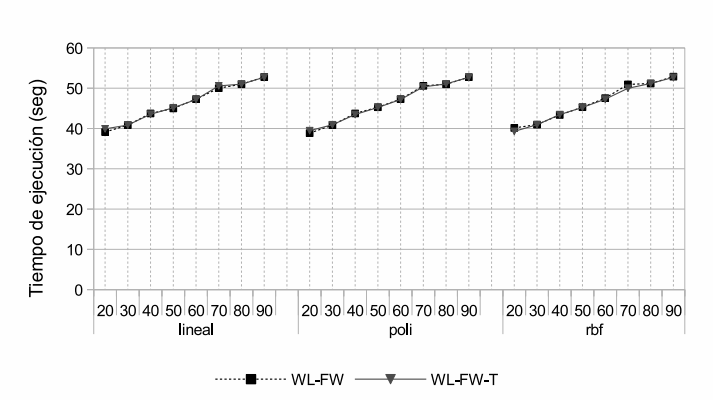


(b) Número de grafos **N**.

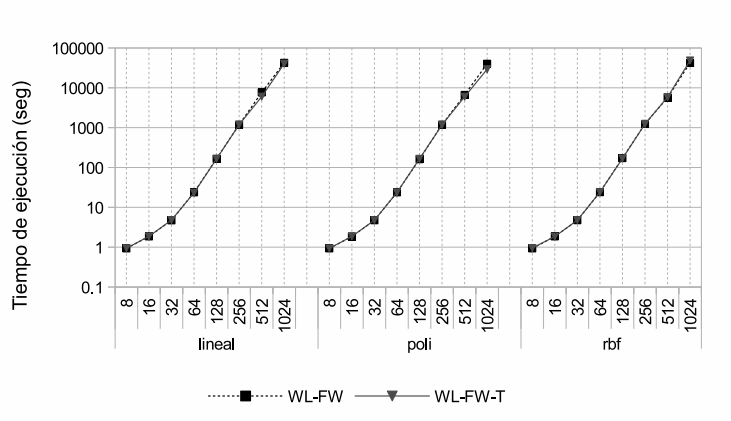


(c) Tamaño de grafos **n**.

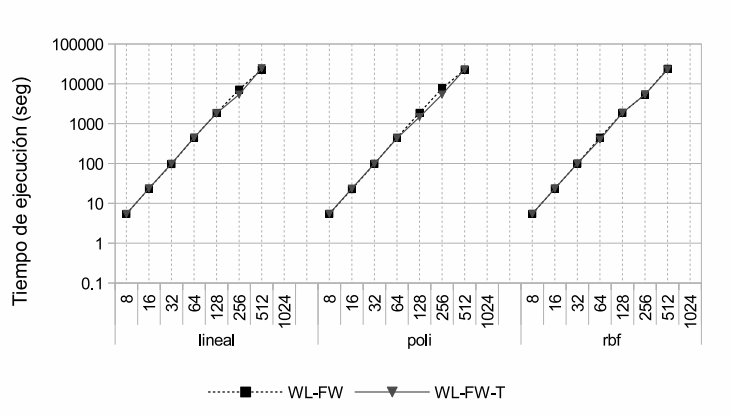
Figura 54. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos (**N** = 64) y nodos por grafo (**n** = 32), (b) el número de grafos **N**, para un número de nodos por grafo fijo (**n** = 16) y densidad de aristas (**c** = 50%) y (c) tamaño de grafos **n**, para un número fijo de grafos (**N** = 64) y densidad de aristas (**c** = 50%). Kernel de *sub-árboles* considerando aristas y sus variantes (*WLE-lineal*, *WLE-poli* y *WLE-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos considerando aristas y sus variantes (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*).



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 55. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variantes (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podados* y sus variantes (*WL-FW-T-lineal*, *WL-FW-T-poli* y *WL-FW-T-rbf*).

Como podemos ver en la Figura 55 todos los valores en los tres experimentos: *densidad*, *número* y *tamaño* (Figuras 55(a), 55(b) y 55(c), respectivamente) de los métodos kernel *WL-FW* y *WL-FW-T* son parecidos entre si.

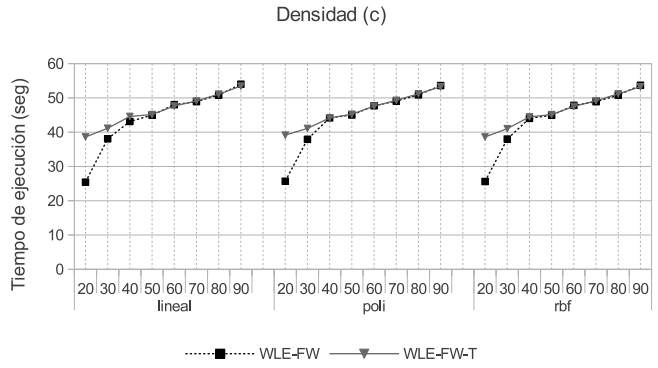
A.1.2.5 Kernel de subárboles considerando aristas sobre grafos de caminos más cortos *podados* (WLE-FW-T)

De la misma manera, la aplicación del *podado* de los grafos de entrada del kernel *WLE-FW* (llamado *WLE-FW-T*) tiene casi el mismo efecto nulo en comparación de su versión sin *podar*.

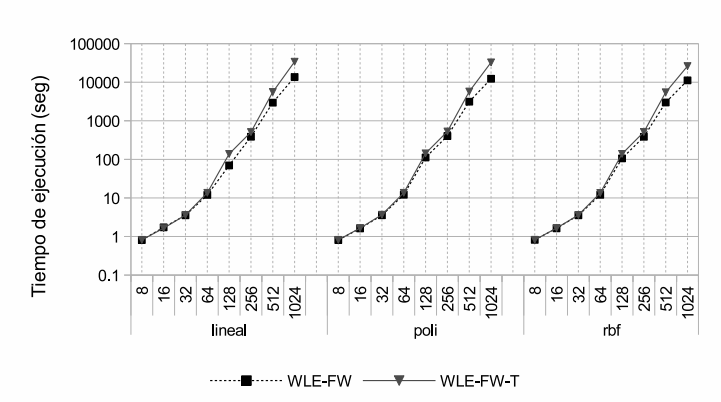
Para el caso en que la variable es la *densidad* (Figura 56(a)) los valores son en su mayoría igual de cercanos entre si que los obtenidos en el experimento de la Figura 55(a), donde la única diferencia es que en el primer caso (con una densidad de 20%) el kernel sin podar *WLE-FW* es más eficiente que su contraparte *WLE-FW-T*.

Cuando se toma el *número* de grafos de entrada como variable (Figura 56(b)) tenemos que ambos métodos kernel tienen resultados comparables y con la misma tendencia. Solo en los últimos dos valores es cuando se empiezan a "despegar" a favor del kernel *WLE-FW* indicando que probablemente para un número mayor de grafos sea mejor evitar el paso de *podarlos*.

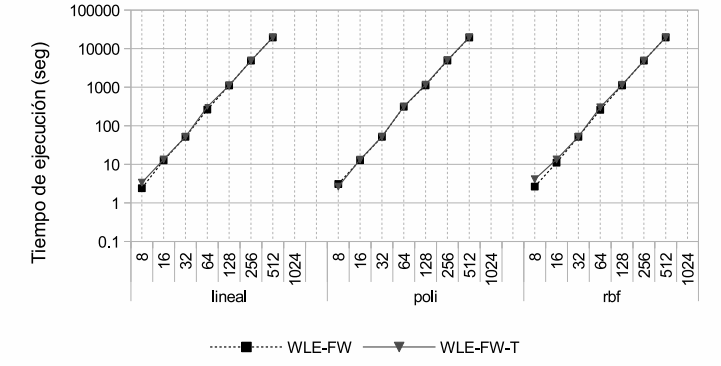
Por último, cuando lo que cambia es el *tamaño* de los grafos de entrada, se puede ver que en este caso (Figura 56(c)) es difícil distinguir cual de los dos obtiene un mejor resultado debido a la similitud de sus resultados.



(a) Densidad de grafos **c**.



(b) Número de grafos **N**.



(c) Tamaño de grafos **n**.

Figura 56. Tiempo de cálculo (*seg.*) del método kernel con respecto a: (a) la densidad de grafos **c**, para un número fijo de grafos ($\mathbf{N} = 64$) y nodos por grafo ($\mathbf{n} = 32$), (b) el número de grafos **N**, para un número de nodos por grafo fijo ($\mathbf{n} = 16$) y densidad de aristas ($\mathbf{c} = 50\%$) y (c) tamaño de grafos **n**, para un número fijo de grafos ($\mathbf{N} = 64$) y densidad de aristas ($\mathbf{c} = 50\%$). Kernel de *sub-árboles* sobre grafo de caminos más cortos considerando aristas y sus variantes (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podados* considerando aristas y sus variantes (*WLE-FW-T-lineal*, *WLE-FW-T-poli* y *WLE-FW-T-rbf*).

Tabla VI. Tiempo de cálculo de la totalidad de los métodos kernel evaluados con respecto a la densidad de los grafos de entrada (c): 20, 30, 40, 50, 60, 70, 80, 90 %.

Kernel \ Densidad	20	30	40	50	60	70	80	90
iSP-lineal	21.2	21.9	22.4	22.75	22.76	23.01	22.82	22.07
iSP-poli	60.4	62.52	62.91	64.55	67.22	68.76	69.65	69.41
iSP-rbf	60.56	62.66	63.07	66.1	67.43	68.93	69.91	69.57
iSP-T-lineal	**8.42**	**8.87**	**8.69**	**8.79**	**8.59**	**8.41**	**8.23**	**7.75**
iSP-T-poli	11.04	11.37	11.26	11.46	11.21	11.03	10.83	10.4
iSP-T-rbf	10.95	11.36	11.24	11.37	11.18	11.02	10.82	10.39
SP-lineal	23.31	23.39	23.47	22.61	23.25	22.34	22.92	22
SP-poli	70.96	69.92	69.47	69.4	69.39	69.51	68.45	68.79
SP-rbf	72	71	70.55	68.99	69.07	70.44	69.53	69.83
SP-T-lineal	8.43	8.91	8.83	8.9	8.61	8.42	8.23	7.76
SP-T-poli	11.21	11.63	11.59	11.55	11.36	11.18	11.07	10.52
SP-T-rbf	11.31	11.71	11.61	11.7	11.47	11.27	11.15	10.63
SPC-lineal	248.26	243.93	247.67	244.1	229.42	226.14	281.62	308.67
SPC-poli	241.01	248.03	248.03	238.06	228.85	225.48	281.36	304.5
SPC-rbf	245.63	246.57	249.66	238.93	232.64	222.72	283.18	310.96
SPC-T-lineal	8.96	9.82	10.02	9.45	9.85	9.66	8.67	8.39
SPC-T-poli	8.9	9.83	10.03	9.41	9.25	9.66	9.29	8.41
SPC-T-rbf	8.93	9.83	10.03	9.42	9.87	9.57	9.23	8.41
WL-FW-lineal	39.12	40.84	43.73	44.98	47.28	50.01	50.98	52.78
WL-FW-poli	38.84	40.85	43.73	45.28	47.28	50.59	50.97	52.74
WL-FW-rbf	40.14	40.96	43.36	45.29	47.54	50.94	51.16	52.86
WL-FW-T-lineal	39.91	40.85	43.53	45.14	47.18	50.57	51	52.72
WL-FW-T-poli	39.45	40.92	43.47	45.21	47.19	50.32	51.07	52.69
WL-FW-T-rbf	39.29	40.93	43.44	45.22	47.27	49.98	51.17	52.69
WL-lineal	30.83	32.87	34.95	37.25	39.18	41.37	44.11	45.51
WL-poli	30.84	32.87	34.85	37.46	40.23	41.36	43.88	45.51
WL-rbf	31.03	32.96	35.14	37.35	40.65	41.39	43.72	45.68
WLE-FW-lineal	25.42	38.07	43.12	44.96	48.03	48.91	50.83	53.99
WLE-FW-poli	25.61	38	44.04	44.93	47.83	48.89	50.8	53.69
WLE-FW-rbf	25.69	37.92	44.18	45.01	47.71	49.06	50.88	53.6
WLE-FW-T-lineal	38.6	41.15	44.6	45.14	47.64	49.09	51.1	53.45
WLE-FW-T-poli	38.6	41.02	44.46	45.13	47.51	49.18	51.18	53.24
WLE-FW-T-rbf	39.12	41.1	44.15	45.24	47.6	49.31	51.2	53.26
WLE-lineal	30.99	32.95	34.9	37.2	40.3	41.13	43.36	46.2
WLE-poli	31.13	32.91	34.9	37.03	40.3	41.21	43.44	46.84
WLE-rbf	31	33.19	35.12	37.14	40.42	41.25	43.56	46.93

Tabla VII. Tiempo de cálculo de la totalidad de los métodos kernel evaluados con respecto al número de grafos de entrada (n): 8, 16, 32, 64, 128, 256, 512 y 1024 grafos.

Kernel \ Número	8	16	32	64	128	256	512	1024
iSP-lineal	0.17	0.37	0.85	2.16	6.23	19.95	70.45	261.51
iSP-poli	0.22	0.55	1.53	4.83	16.76	61.68	232.28	886.59
iSP-rbf	0.23	0.56	1.53	4.84	16.79	61.43	233.86	903.44
iSP-T-lineal	**0.16**	**0.32**	**0.65**	**1.39**	**3.05**	7.45	20.05	60.41
iSP-T-poli	**0.16**	0.33	0.76	1.78	4.56	13.02	42.21	148.58
iSP-T-rbf	**0.16**	0.33	0.75	1.76	4.53	12.91	41.53	146.08
SP-lineal	0.18	0.38	0.88	2.3	6.77	22.06	78.71	301.1
SP-poli	0.24	0.58	1.7	5.47	19.26	71.58	275.94	1116.5
SP-rbf	0.23	0.6	1.81	5.56	19.64	73.17	280.71	1073.7
SP-T-lineal	0.15	**0.32**	0.67	1.44	3.2	7.64	20.63	62.7
SP-T-poli	0.17	0.34	0.77	1.83	4.78	13.94	45.78	162.6
SP-T-rbf	0.16	0.34	0.77	1.83	4.76	13.81	45.36	161.48
SPC-lineal	0.95	1.65	3.72	15.2	121.95	859.27	3015	29748
SPC-poli	0.84	1.65	3.73	15.64	122.56	690.61	4743.3	27813
SPC-rbf	0.91	1.65	3.74	15.32	126.02	771.32	4257.3	31157
SPC-T-lineal	0.27	0.46	0.92	1.91	3.67	**7.31**	**15.17**	35.86
SPC-T-poli	0.23	0.47	0.9	1.83	3.67	7.33	15.22	**32.32**
SPC-T-rbf	0.24	0.46	0.92	1.84	3.69	7.45	15.62	38.56
WL-FW-lineal	0.93	1.86	4.7	23.75	165.3	1187.1	7657.3	42514
WL-FW-poli	0.93	1.85	4.72	23.76	161.27	1188.3	6647.9	40276
WL-FW-rbf	0.93	1.86	4.74	23.92	171.66	1240.6	5693.3	42396
WL-FW-T-lineal	0.93	1.87	4.74	23.66	165.34	1179.8	5958.3	29198
WL-FW-T-poli	0.95	1.85	4.75	24.03	168.33	1239.3	5817.6	48486
WL-FW-T-rbf	0.68	1.39	2.96	15.41	116.31	725.9	5410.5	41862
WL-lineal	0.66	1.33	2.96	12.39	116.81	775.5	5499	39579
WL-poli	0.65	1.33	2.98	12.35	120.33	760.68	5756	27274
WL-rbf	0.81	1.73	3.55	11.97	69.42	385.79	2954.1	13624
WLE-FW-lineal	0.82	1.63	3.55	12.04	105.94	385.1	2979.7	11153
WLE-FW-poli	0.81	1.62	3.56	12.11	111.85	402.88	3132.1	12316
WLE-FW-rbf	0.82	1.64	3.61	13.58	138.15	510.61	5583.5	34042
WLE-FW-T-lineal	0.94	1.87	4.79	23.24	165.19	1186.5	6003	40229
WLE-FW-T-poli	0.82	1.65	3.62	13.46	138.43	513.01	5486.2	26178
WLE-FW-T-rbf	0.82	1.65	3.67	13.57	143.1	528.97	5745.9	32702
WLE-lineal	0.65	1.32	2.96	12.76	116.78	676.05	5428.2	25353
WLE-poli	0.65	1.33	2.95	12.34	116.79	659.73	5432.2	25316
WLE-rbf	0.67	1.33	2.96	12.3	120.59	514.82	5679.1	33414

Tabla VIII. Tiempo de cálculo (segundos) de la totalidad de los métodos kernel evaluados con respecto al tamaño de los grafos de entrada (N): 8, 16, 32, 64, 128, 256, 512 y 1024 nodos.

Kernel \ Tamaño	8	16	32	64	128	256	512	1024
iSP-lineal	**0.29**	2.19	21.85	278.03	4390.7	72883	n/a	n/a
iSP-poli	0.45	4.69	65.02	1059	18573	n/a	n/a	n/a
iSP-rbf	0.46	4.71	64.95	1055	19028	n/a	n/a	n/a
iSP-T-lineal	0.26	**1.4**	8.98	54.59	329.75	2071.3	13980	n/a
iSP-T-poli	0.31	1.76	11.41	73.1	475.49	3195.7	n/a	n/a
iSP-T-rbf	0.3	1.73	11.4	73.27	478.33	3207.4	n/a	n/a
SP-lineal	0.31	2.35	22.61	267.58	3614.2	55265	n/a	n/a
SP-poli	0.54	5.43	67.94	970.19	14981	n/a	n/a	n/a
SP-rbf	0.52	5.39	68.2	985.39	15316	229590	n/a	n/a
SP-T-lineal	0.27	1.42	**8.62**	**52.91**	**323.68**	2030	13721	n/a
SP-T-poli	0.33	1.81	11.6	73.73	476.14	3202.4	22810	n/a
SP-T-rbf	0.32	1.82	11.75	73.77	478.08	3202.2	22889	n/a
SPC-lineal	1.67	12.31	192.7	2521.5	69042	n/a	n/a	n/a
SPC-poli	1.74	14.88	253.24	4096.4	64927	n/a	n/a	n/a
SPC-rbf	1.68	11.47	263.27	2577.2	69397	n/a	n/a	n/a
SPC-T-lineal	0.5	1.86	9.28	50.97	294.55	**1818**	11641	n/a
SPC-T-poli	0.49	1.81	9.3	51.03	294.85	1841.6	**10950**	n/a
SPC-T-rbf	0.49	1.94	9.3	50.93	294.63	1877.2	10975	n/a
WL-FW-lineal	5.44	23.37	96.8	443.51	1856.8	7168.2	22716	n/a
WL-FW-poli	5.43	22.69	97.37	444.25	1869.7	7620.2	22500	n/a
WL-FW-rbf	5.44	23.48	99.36	446.69	1871.1	5376.8	23870	n/a
WL-FW-T-lineal	5.4	23.61	98.86	445.27	1821.9	5376.5	24598	n/a
WL-FW-T-poli	5.47	23.47	99.01	430.34	1461.8	5380.7	23019	n/a
WL-FW-T-rbf	5.47	23.44	99.39	398.11	1876.4	5382.6	22876	n/a
WL-lineal	3.5	12.85	42.67	220.03	824.7	3080.6	8723.1	35408
WL-poli	2.82	13.4	42.7	213.43	824.87	3353.6	8747.5	34727
WL-rbf	3.32	8.61	43.61	205.32	829.56	3381.2	8731.5	34784
WLE-FW-lineal	2.4	12.71	51.79	263.08	1109.9	4898.2	19459	n/a
WLE-FW-poli	2.65	11.06	51.72	258.76	1108.8	4893.1	19442	n/a
WLE-FW-rbf	3.09	12.79	51.82	311.67	1112.9	4910.3	19416	n/a
WLE-FW-T-lineal	3.35	13.4	51.96	310.26	1107	4836.9	19569	n/a
WLE-FW-T-poli	4.11	13.37	51.99	300.32	1155.4	4801.4	19703	n/a
WLE-FW-T-rbf	2.64	13.23	52.12	314.16	1162.9	5034.9	19665	n/a
WLE-lineal	2.68	8.61	42.9	218.01	824.77	3113.2	8624.2	34100
WLE-poli	2.36	8.53	42.97	205.69	822.67	3078.7	8563.3	34043
WLE-rbf	2.47	9.67	43.02	209.61	825.09	3093.7	8620.5	**34038**

A.2 Resultados de experimentos con esquemas de etiquetado

En esta sección se presenta el resto de los resultados de los experimentos con grafos etiquetados de acuerdo a esquemas mostrados en el capítulo IV.

Primero se muestran lo resultados obtenidos con las variantes del kernel de caminos más cortos y después los resultados correspondientes a los experimentos realizados con las variantes del kernel de subárboles.

A.2.1 Variantes de kernel de caminos más cortos (SP)

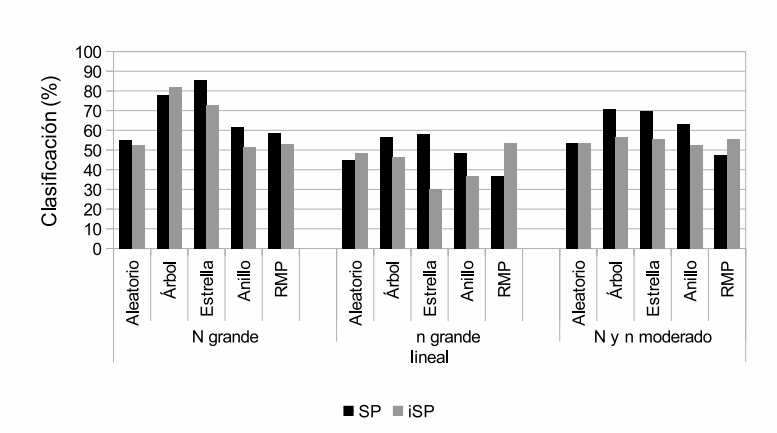
Kernel de caminos más cortos invertido (iSP)

Para la comparación entre las tres variantes del kernel de *caminos más cortos* (*SP-lineal*, *SP-poli* y *SP-rbf*) y el kernel de *caminos más cortos invertido* y sus tres variaciones (*iSP-lineal*, *iSP-poli* y *iSP-rbf*) tenemos la Figura 57.

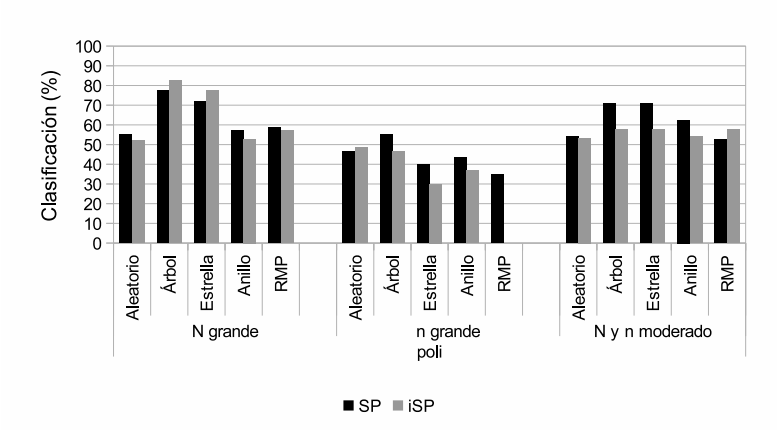
Para las variantes que utilizan el kernel *lineal*, (Figura 57(a)) tenemos que la mayoría de los resultados favorecen al kernel *SP-lineal* pero solo para el grupo "N grande", con topología *estrella*, se tiene un resultado superior al 80%, el resto es inferior o cercano al 70%. El otro resultado superior al 80% es del mismo grupo pero resultante del kernel *iSP* (su único valor relevante).

En el segundo caso (kernel *polinomial*) se puede apreciar (Figura 57(a)) que la mayor parte de los valores son similares a los de la Figura 57(a) y que los valores más significativos pertenecen al kernel *iSP-poli* que se aplica sobre grafos con topología de *árbol* y *estrella* en el primer grupo del experimento (*"N grande"*) ya que obtuvieron un porcentaje de clasificación de 82.5% y 77.67% respectivamente.

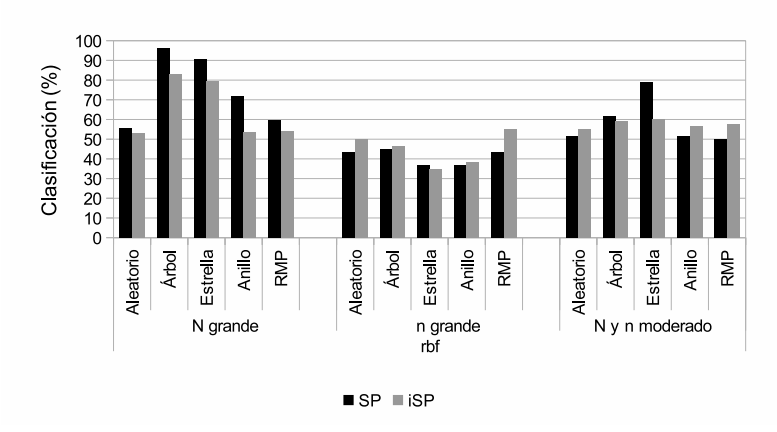
Por último, el kernel *SP-rbf* fue el que tuvo resultados más relevantes en el primer grupo (*"N grande"*) sobre grafos con topología de *árbol* y *estrella* (96.17% resp. 90.67%) y en el tercer grupo (*"N y n moderado"*) sobre grafos con topología de *estrella* con 79.17% de clasificación. Para el segundo grupo (*"n grande"*) se tuvieron resultados menores o muy



(a) Esquemas de etiquetado variación *lineal*



(b) Esquemas de etiquetado variación *polinomial*



(c) Esquemas de etiquetado variación *rbf*

Figura 57. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de caminos más cortos y sus variaciones (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos invertido y sus variaciones (*iSP-lineal*, *iSP-poli* e *iSP-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

cercanos al 50%.

Kernel de caminos más cortos *podados* (SP-T)

En el experimento realizado para evaluar el impacto del proceso de *podado* sobre los grafos de entrada del kernel de *caminos más cortos* (*SP-T*), se pudo constatar que la variación ya mencionada resulta en una ventaja sobre su contraparte original del método kernel (*SP*).

En la Figura 58(a) se puede ver como el kernel *SP-T* es el que obtuvo mejores resultados (mayores de 70%) y para el resto se tuvo resultados muy pobres, menores o muy cerca del 60% en el mejor de los casos.

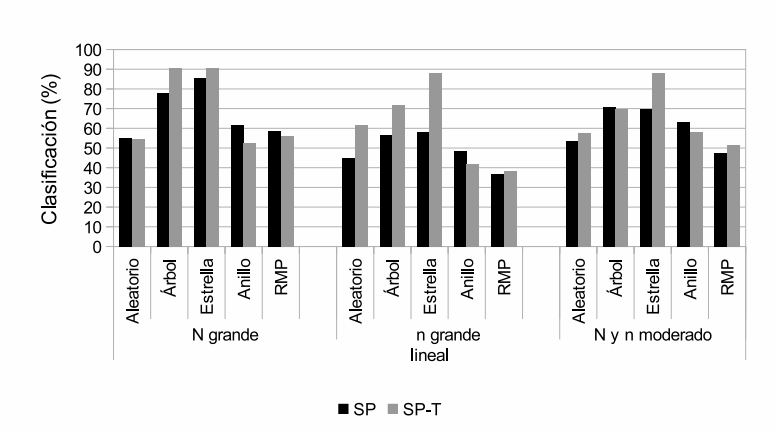
Para la Figura 58(b) se repite el mismo comportamiento, siendo *SP-T* el que mejores resultados obtuvo. Además de esto, podemos ver como todos los resultados por lo general son similares excepto los resultados del grupo *"n grande"* en donde disminuyó notáblemente el promedio del kernel *SP*.

Los resultados de las variantes que utilizan el kernel *RBF* (Figura 58(c)) vieron una mejora general para el kernel *SP* en los grupos 1 y 3 mientras que el grupo 2 mantuvo el mismo promedio pobre que se obtuvo con las variantes que utlizan el kernel *polinomial*. A pesar de esto, el kernel *SP-T* sigue teniendo mejores resultados sobre todo en el grupo 1 en donde se obtuvo el arriba de 90% para la topología *árbol* con ambos métodos kernel.

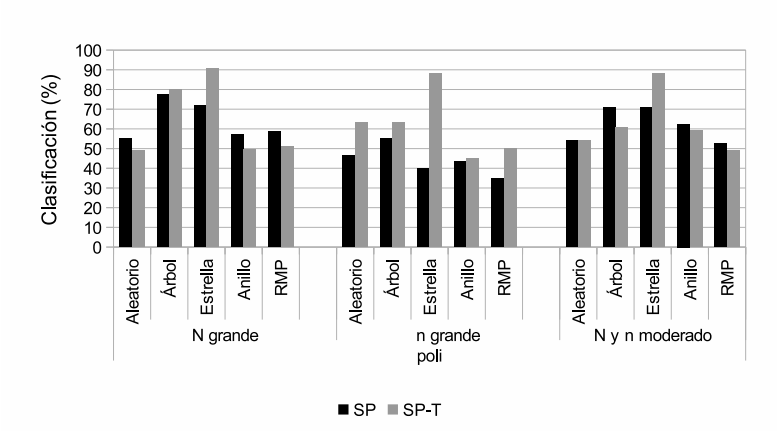
Kernel de caminos más cortos invertido *podado* (iSP-T)

Cuando evaluamos el comportamiento del kernel *podado iSP-T* a comparación de su contraparte *iSP*, (Figura 59) nos podemos dar cuenta que el proceso de *podar* resulta exitoso para obtener un mejor porcentaje de clasificación de grafos etiquetados por los esquemas implementados con respecto a la versión normal del kernel de *caminos más cortos invertido*.

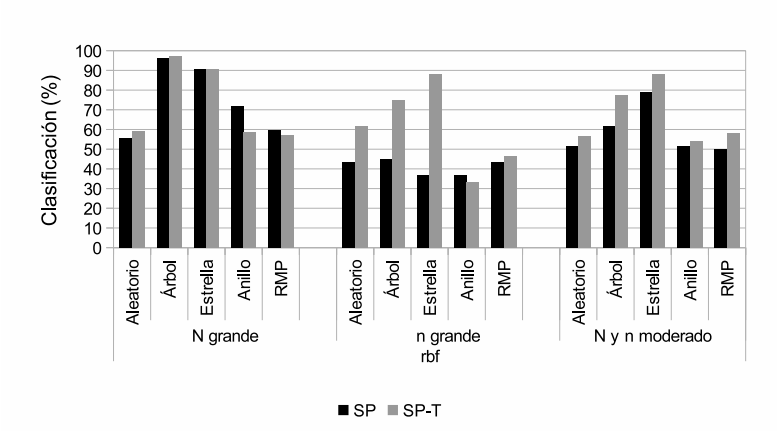
Estos resultados coinciden con el comportamiento del experimento de la Figura 58 en donde también resultó en una mejora pero esta vez sobre el kernel de *caminos más cortos* normal.



(a) Esquemas de etiquetado variación *lineal*

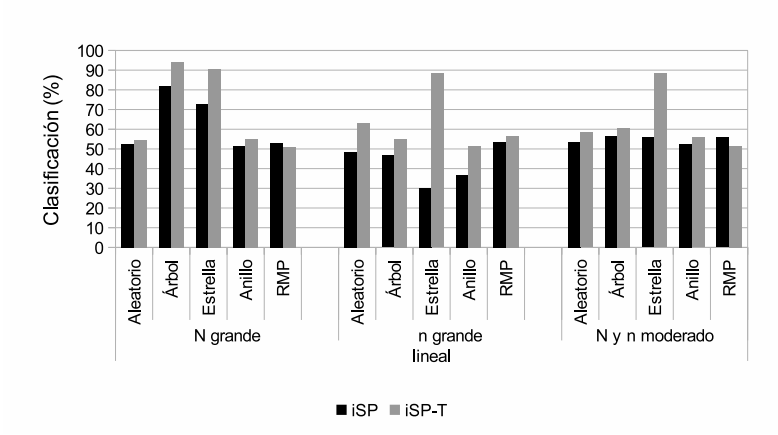


(b) Esquemas de etiquetado variación *polinomial*

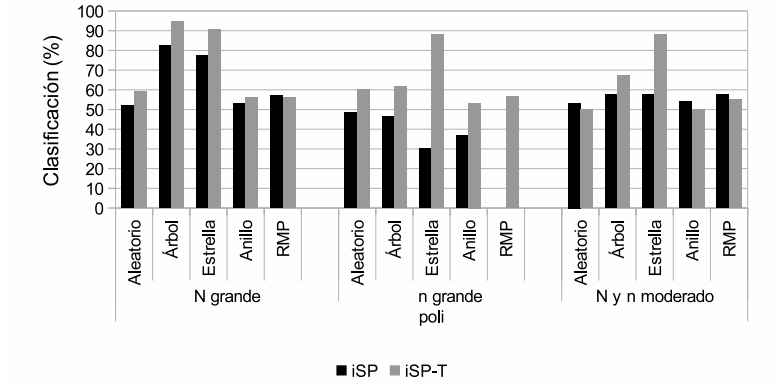


(c) Esquemas de etiquetado variación *rbf*

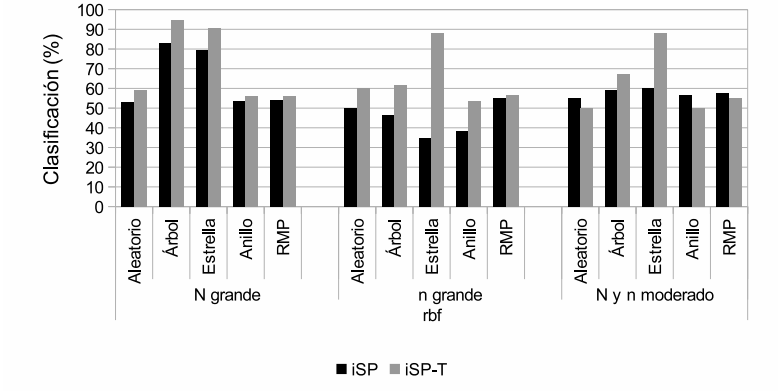
Figura 58. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de caminos más cortos y sus variaciones (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de caminos más cortos *podado* y sus variaciones (*SP-T-lineal*, *SP-T-poli* y *SP-T-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).



(a) Esquemas de etiquetado variación *lineal*



(b) Esquemas de etiquetado variación *polinomial*



(c) Esquemas de etiquetado variación *rbf*

Figura 59. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de caminos más cortos invertido y sus variaciones (*iSP-lineal*, *iSP-poli* e *iSP-rbf*) y kernel de caminos más cortos invertido *podado* y sus variaciones(*iSP-T-lineal*, *iSP-T-poli* e *iSP-T-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

Si bien, el proceso de *podado* no representa un aumento demasiado substancial, si aumenta el porcentaje de clasificación correcta en la mayoría de las instancias probadas por un promedio de alrededor de 10% y en casos extremos por más de 40% sobre su contraparte.

En las Figuras 59(a), 59(b) y 59(c) podemos ver como *iSP-T* sobresale con respecto a *iSP* en casi todos los casos.

Un detalle importante es el hecho de que por lo general los resultados son bajos con excepción de los correspondientes a los grafos con topología *árbol* y *estrella* sobre todo en el grupo 1 de las tres variantes (*lineal*, *polinomial* y *rbf*).

Kernel de conteo de caminos más cortos (SPC)

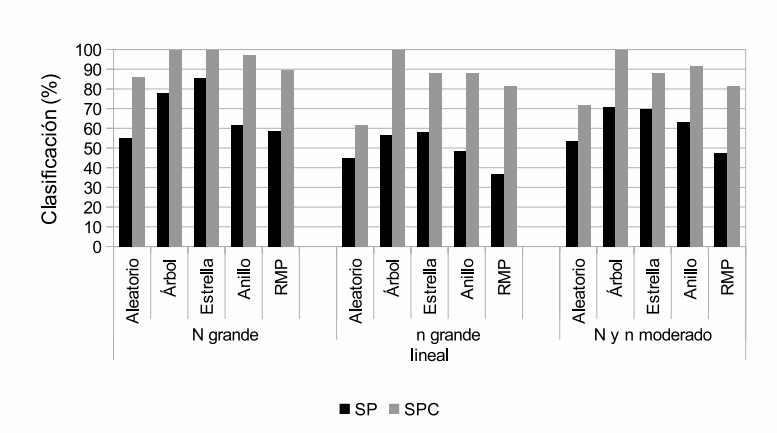
Quizá la mejora más significativa en todos los casos fue la que presentó el kernel de *conteo de caminos más cortos* (*SPC*) sobre el kernel de *caminos más cortos* original (*SP*).

En la Figura 60 nos podemos dar cuenta de manera inmediata cómo *SPC* mejora virtualmente todos los resultados obtenidos por *SP*. Incluso se puede ver cómo para el grupo 2, que en casi todos los experimentos es el de peor desempeño, la mayoría de los resultados son mayores a 80% (excepto para *SPC-rbf* en 60(c)).

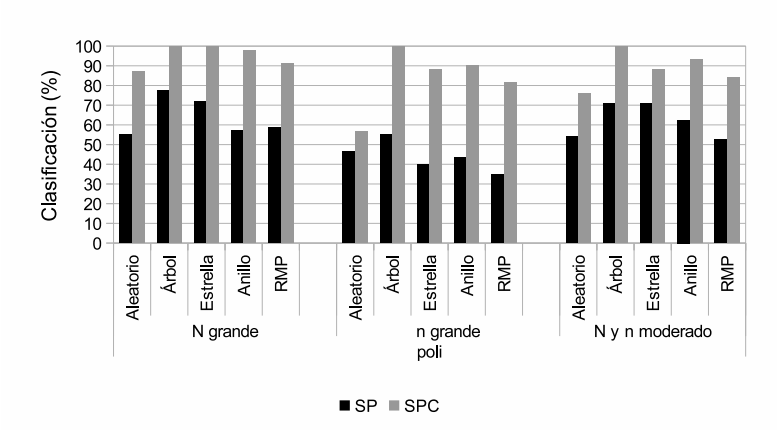
Cabe destacar que el kernel *SPC* mejora tanto el desempeño del kernel *SP* que su variante que utiliza el kernel *polinomial* (*SPC-poli*) obtiene 4 de los 5 resultados máximos para cualquier kernel cuando el número de grafos es el que se aumenta considerablemente: *Aleatorio*: 87%, *Árbol*: 100%, *Estrella*: 100%, *Anillo*: 97.67% y *Red de mundo pequeño*: 91.33%. Solo el resultado de la topología *anillo* no es el máximo porcentaje (el máximo es 99.83%), sin embargo se encuentra muy cerca.

Kernel de conteo de caminos más cortos *podado* (SPC-T)

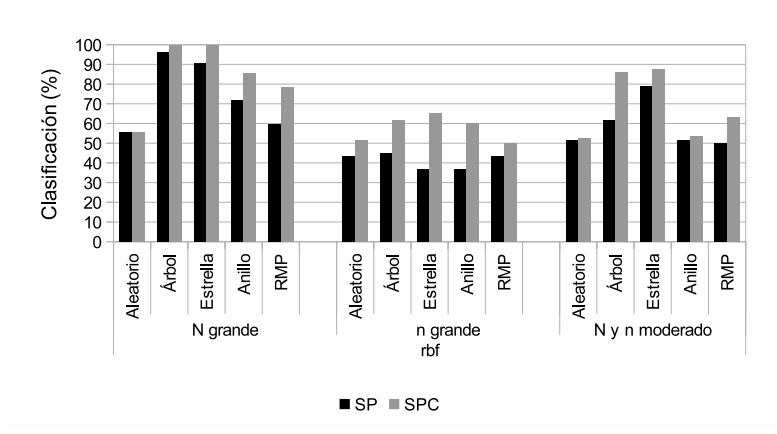
Una vez visto el buen desempeño obtenido por *SPC* en comparación a *SP* para la clasificación de grafos por esquemas de etiquetado, se evalúa el desempeño de su versión *podada SPC-T* esperando tener resultados similares pues es una versión pensada en tener mejor desempeño en tiempo de cálculo que su contraparte.



(a) Esquemas de etiquetado variación *lineal*



(b) Esquemas de etiquetado variación *polinomial*



(c) Esquemas de etiquetado variación *rbf*

Figura 60. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de caminos más cortos y sus variaciones (*SP-lineal*, *SP-poli* y *SP-rbf*) y kernel de conteo de caminos más cortos y sus variaciones(*SPC-lineal*, *SPC-poli* y *SPC-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

Después de analizar los resultados (Figura 61), se puede concluir que, aunque los resultados son competitivos, no resultan ser superiores al kernel *SPC*.

Además de esto, existen casos en los que la distancia entre los resultados de los métodos kernel *SPC* y *SPC-T* favorecen por más de 20% al primero especialmente cuando se trata con grafos con topología tipo *anillo*. Sin embargo, aunque su comportamiento es desfavorable con respecto a su contraparte, todavía resulta una mejora sobre los resultados arrojados por el kernel original *SP* para la clasificación por esquemas de etiquetado.

A.2.2 Variantes de kernel de subárboles (WL)

Kernel de subárboles tomando en cuenta aristas (WLE)

En la Figura 62 se muestra la comparación de resultados de clasificación entre el kernel de *subárboles* y su versión que toma en cuenta aristas *WLE*. Al analizar los resultados, tenemos que la variación en la que se toma en cuenta las aristas del kernel de *subárboles* (*WLE*) resulta una mejora sobre el kernel original ya que mejora visíblemente el desempeño sobre grafos con topología tipo *anillo* y además obtiene resultados similares en el resto.

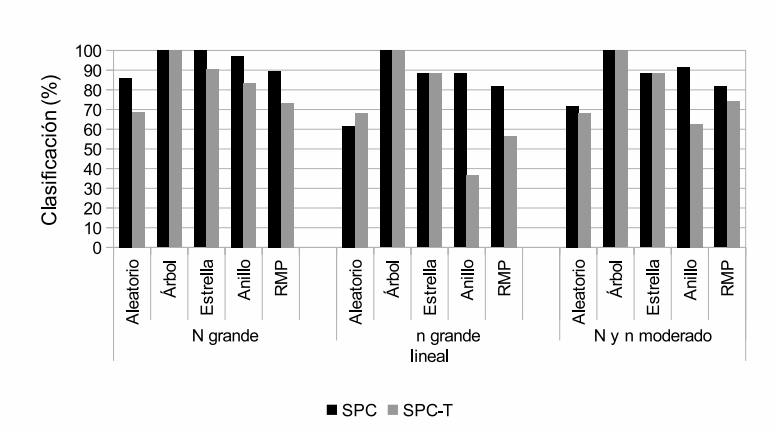
La única instancia en la que *WL* resulta mejor que *WLE* es en el caso de que se estén clasificando grafos con topología tipo *red de mundo pequeño* (*RMP*) ya que obtuvo mejores resultados en dos de los tres grupos en cada una de las variantes del kernel (*WL-lineal*, *WL-poli* y *WL-rbf*).

Kernel de caminos más cortos sobre grafo de caminos más cortos (WL-FW)

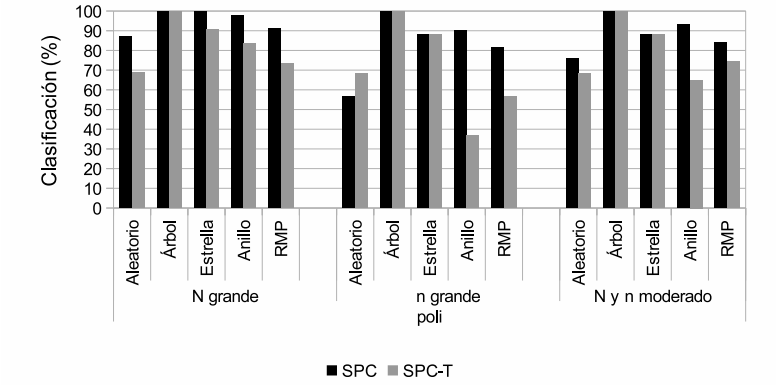
La Figura 63 muestra el impacto que tiene el proceso de transformar los grafos de entrada en grafos de *caminos más cortos* antes de procesarlos con el kernel de *subárboles*, este kernel es conocido como *WL-FW*.

En la Figura podemos ver cómo los resultados arrojados por los kernels *WL* y *WL-FW* son muy similares entre si, es decir, que la transformación aporta poco para mejorar el porcentaje de clasificación, al contrario de lo que se pensaba.

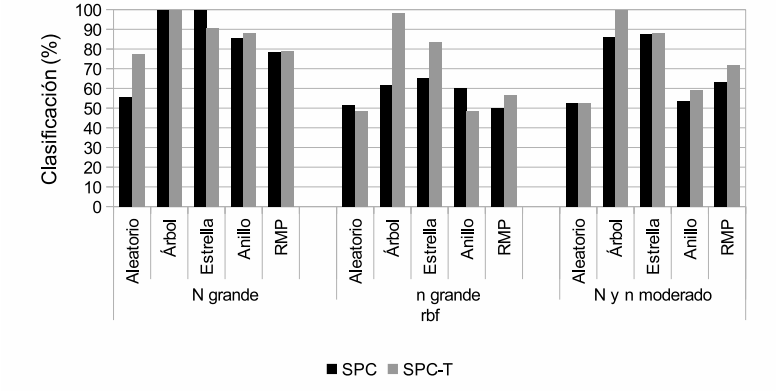
El kernel *WL-FW* obtuvo pocos resultados mayores que su equivalente con el kernel



(a) Esquemas de etiquetado variación *lineal*

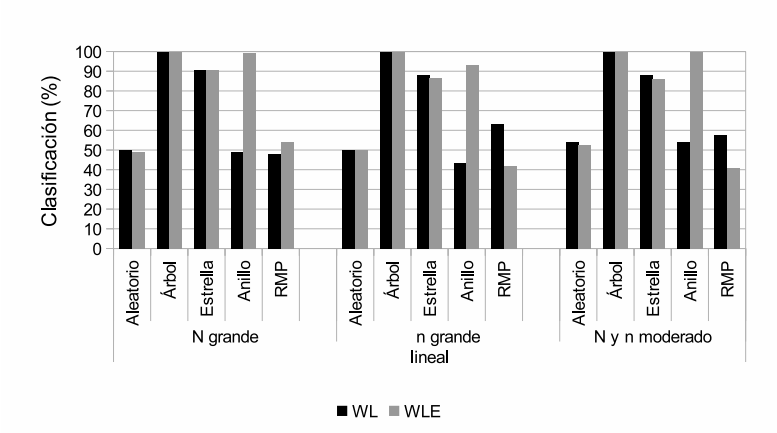


(b) Esquemas de etiquetado variación *polinomial*

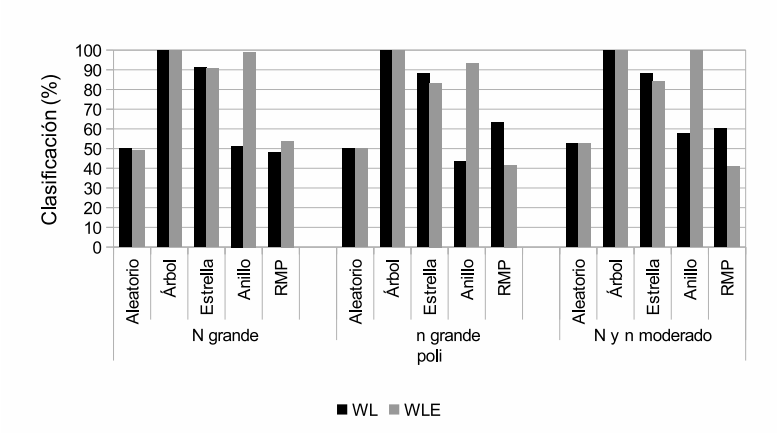


(c) Esquemas de etiquetado variación *rbf*

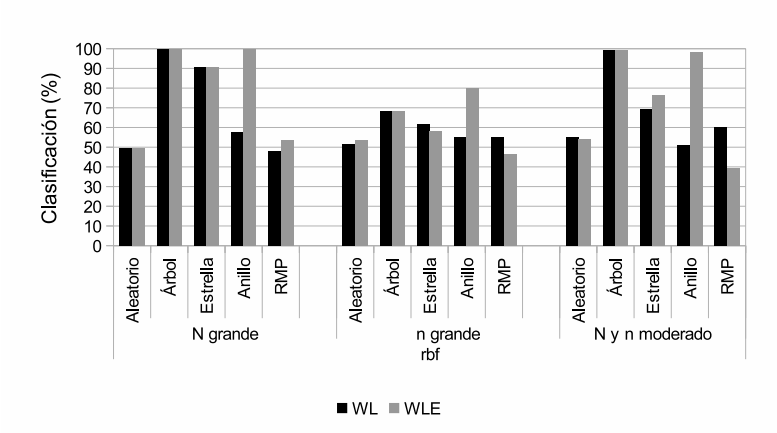
Figura 61. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de *conteo de caminos más cortos* y sus variaciones (*SPC-lineal*, *SPC-poli* y *SPC-rbf*) y kernel de *conteo de caminos más cortos podado* y sus variaciones(*SPC-T-lineal*, *SPC-T-poli* y *SPC-T-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).



(a) Esquemas de etiquetado variación *lineal*

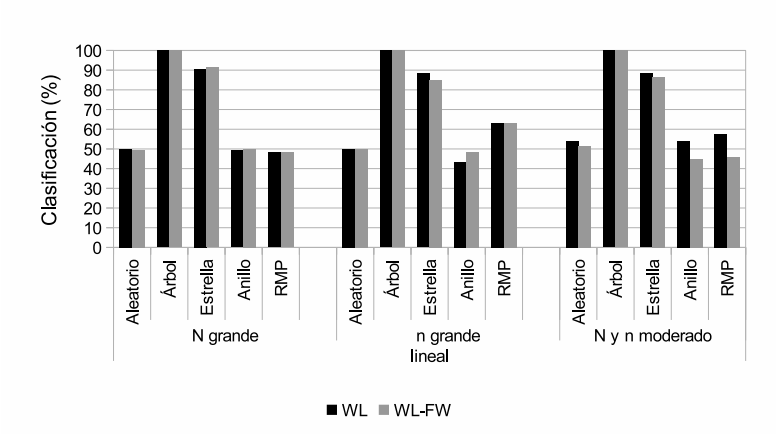


(b) Esquemas de etiquetado variación *polinomial*

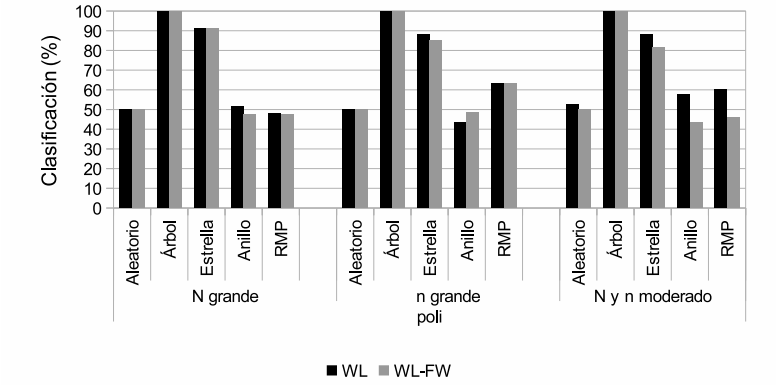


(c) Esquemas de etiquetado variación *rbf*

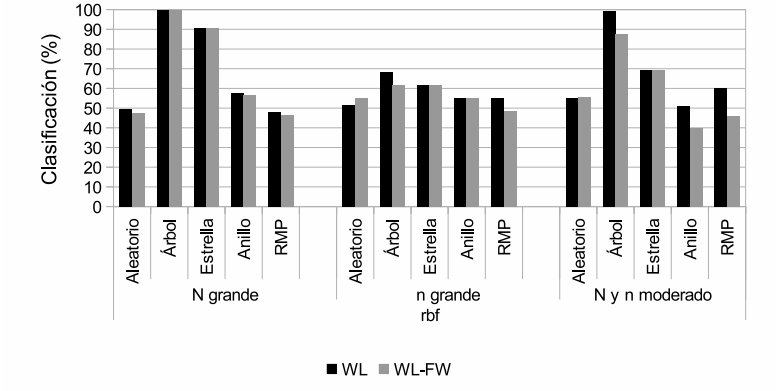
Figura 62. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de sub-árboles y sus variaciones (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de sub-árboles tomando en cuenta aristas y sus variaciones (*WLE-lineal*, *WLE-poli* y *WLE-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).



(a) Esquemas de etiquetado variación *lineal*



(b) Esquemas de etiquetado variación *polinomial*



(c) Esquemas de etiquetado variación *rbf*

Figura 63. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de *sub-árboles* y sus variaciones (*WL-lineal*, *WL-poli* y *WL-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variaciones (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

WL pero la diferencia no es lo suficientemente grande para considerarlos un punto a su favor para ser elegido sobre su contraparte.

Kernel de subárboles tomando en cuenta aristas sobre grafo de caminos más cortos (WLE-FW)

Cuando se evalúa el impacto de la transformación *Floyd-Warshall* sobre los grafos antes de aplicar el kernel de *subárboles* tomando en cuenta aristas (*WLE-FW*) nos podemos percatar que sucede virtualmente lo mismo que sucede con *WL* y *WL-FW* con la excepción de .

Como se puede ver en la Figura 64, los valores arrojados por los kernels *WLE* y *WLE-FW* obtuvieron casi el mismo resultado para la mayoría de los conjuntos de datos, en los tres grupos y con las tres variaciones de kernels (*lineal*, *poli* y *rbf*).

Al igual que para su contraparte *WL*, el proceso de aplicar la transformación de *Floyd-Warshall* aporta muy poco para la mejora en resultados de clasificación lo que lo hace difícil de escoger sobre el original a menos de que tenga mejor desempeño en tiempo de ejecución.

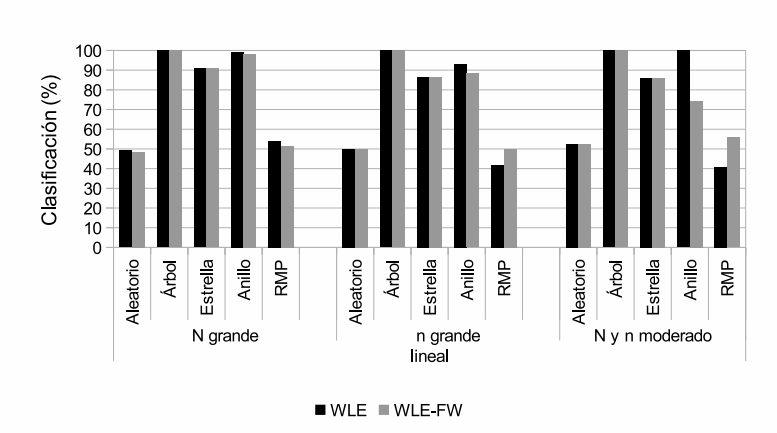
Kernel de subárboles sobre grafo de caminos más cortos *podado* (WL-FW-T)

El propósito de *podar* los grafos antes de aplicar el kernel *WL-FW* es reducir, en la medida de los posible, el tiempo de cálculo del kernel esperando no sacrificar mucho en porcentaje de clasificación.

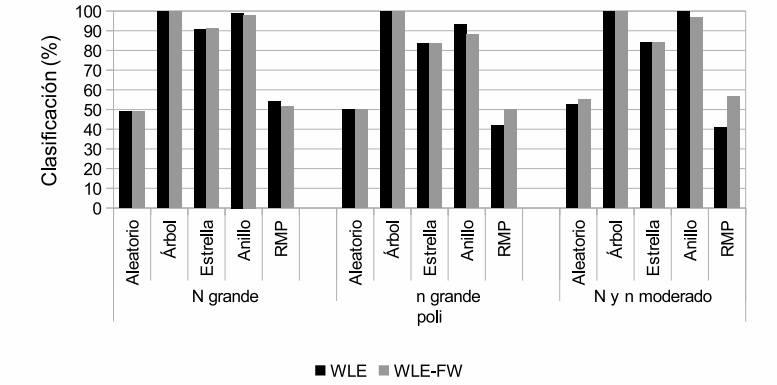
En la Figura 65 se puede ver inmediatemente cómo el proceso de *podar* no tuvo ningún efecto (positivo o negativo) en la clasificación ya que los resultados son exactamente los mismos en para todos conjuntos de datos, los tres grupos y con las tres variantes de kernel (*WL-FW-T-lineal*, *WL-FW-T-poli* y *WL-FW-T-rbf*).

Kernel de subárboles tomando en cuenta aristas sobre grafo de caminos más cortos *podado* (WLE-FW-T)

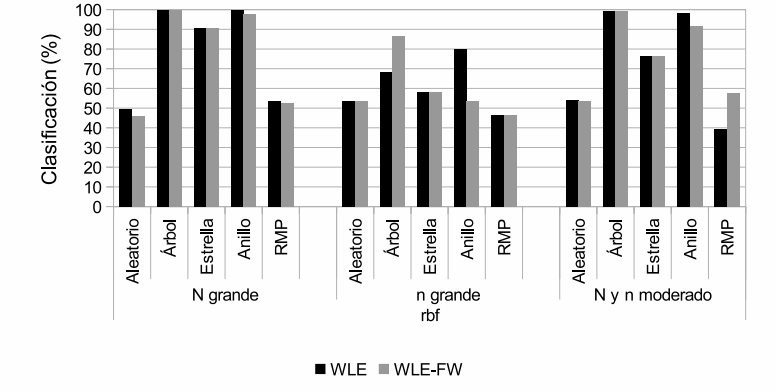
Para la evaluación del impacto de *podar* el kernel *WLE-FW* se puede observar que tampoco ofrece mucho cambio con respecto al original *WL-FW*.



(a) Esquemas de etiquetado variación *lineal*

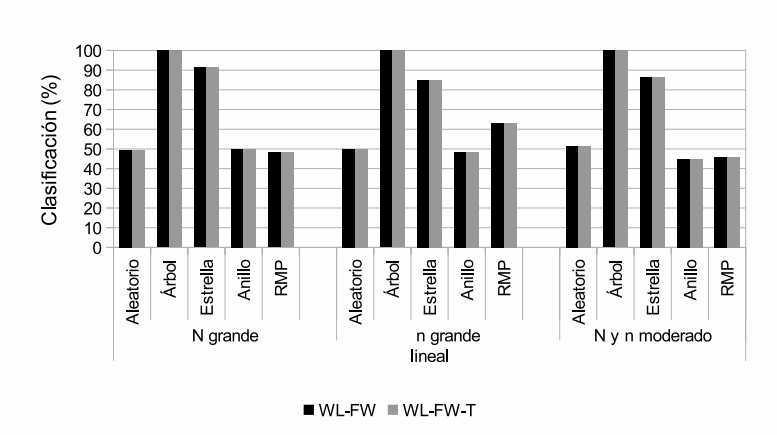


(b) Esquemas de etiquetado variación *polinomial*

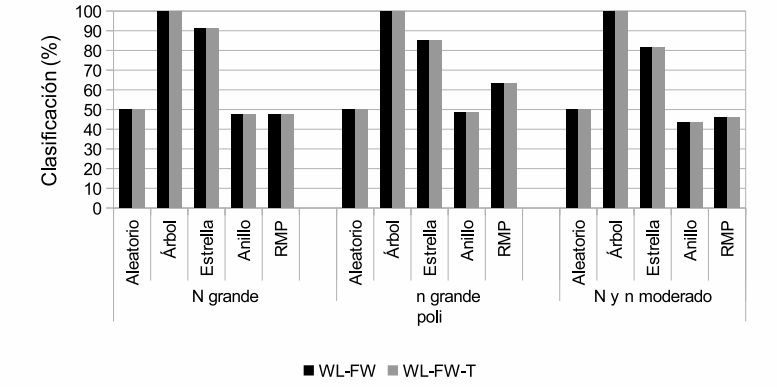


(c) Esquemas de etiquetado variación *rbf*

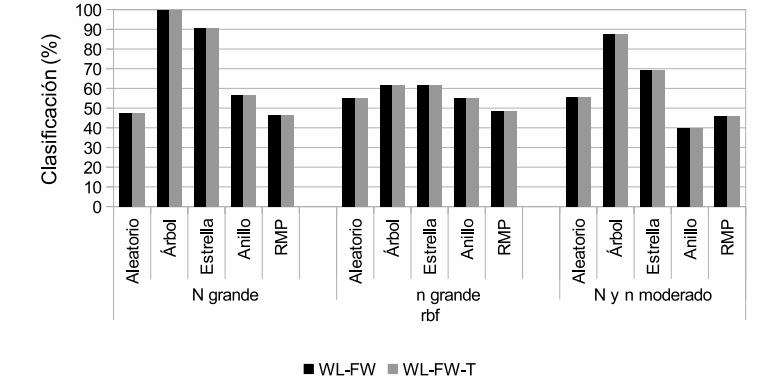
Figura 64. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de *sub-árboles* tomando en cuenta aristas y sus variaciones (*WLE-lineal*, *WLE-poli* y *WLE-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos tomando en cuenta aristas y sus variaciones (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).



(a) Esquemas de etiquetado variación *lineal*

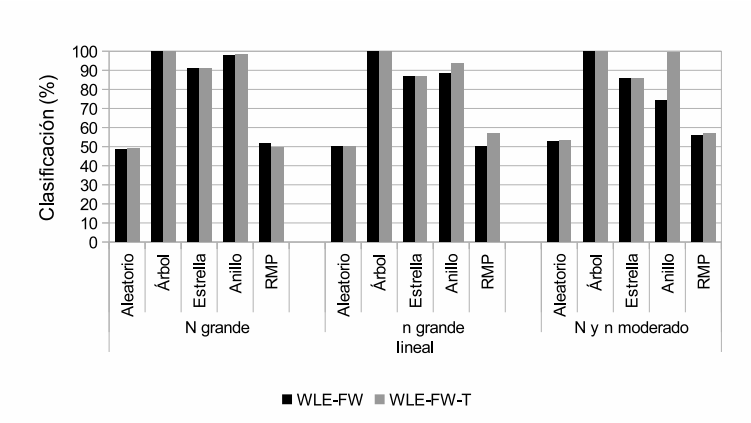


(b) Esquemas de etiquetado variación *polinomial*

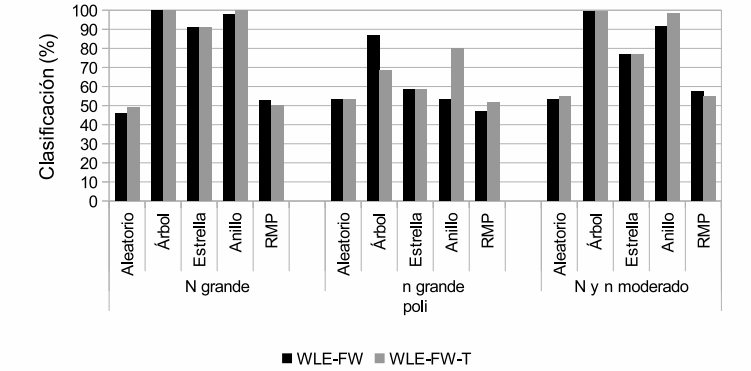


(c) Esquemas de etiquetado variación *rbf*

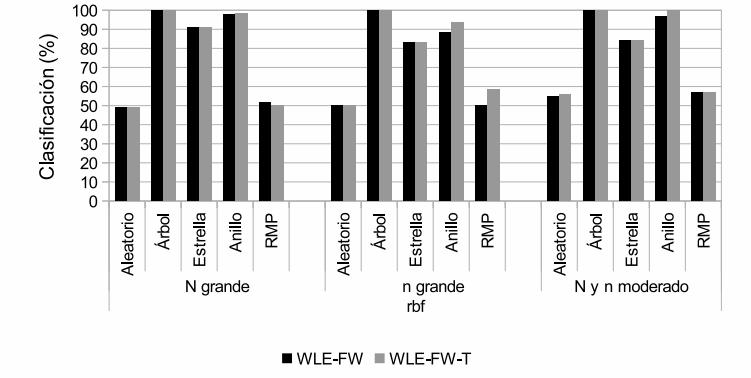
Figura 65. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de *sub-árboles* sobre grafo de caminos más cortos y sus variaciones (*WL-FW-lineal*, *WL-FW-poli* y *WL-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podados* y sus variaciones (*WL-FW-T-lineal*, *WL-FW-T-poli* y *WL-FW-T-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).



(a) Esquemas de etiquetado variación *lineal*



(b) Esquemas de etiquetado variación *polinomial*



(c) Esquemas de etiquetado variación *rbf*

Figura 66. Desempeño de clasificación (%) de grafos por esquemas de etiquetado. Kernel de *sub-árboles* sobre grafo de caminos más cortos tomando en cuenta aristas y sus variaciones (*WLE-FW-lineal*, *WLE-FW-poli* y *WLE-FW-rbf*) y el kernel de *sub-árboles* sobre grafo de caminos más cortos *podados* tomando en cuenta aristas y sus variaciones (*WLE-FW-T-lineal*, *WLE-FW-T-poli* y *WLE-FW-T-rbf*). Conjuntos: N grande (N = 600 y n = 20-30); n grande (N = 60 y n = 60-200); N y n moderado (N = 120 y n = 25-60).

En la Figura 66 se puede observar cómo se tienen resultados muy parecidos a los obtenidos en la comparación de la Figura 65 aunque no exactamente iguales como en ese caso.

Aunque, en la gran mayoría de los resultados los valores son iguales, la variación *podada* (*WLE-FW-T*) es superior en los valores con diferencias significativas lo que la hace una mejor opción sobre *WLE-FW*.

Apéndice B

Conjuntos de datos para experimentos

B.1 Generación de datos

B.1.1 Paquete igraph

Para la generación de los grafos prueba, se seleccionó el paquete *igraph* (Csardi y Tamas Nepusz, 2006) el cual está disponible en una variedad de lenguajes de programación. En este trabajo se utiliza el paquete que está incluido en el lenguaje de programación *R*.

El paquete *igraph* permite generar, manipular y visualizar grafos de distintos tipos y tamaños así como asignar atributos como pesos y colores tanto a los vértices como a las aristas de dichos grafos. Estas características permiten generar una gran variedad de grafos que permitan modelar problemas del mundo real.

B.1.1.1 Almacenamiento de información

Debido a que los grafos se generan con el paquete *igraph* para el lenguaje *R* y los métodos kernels así como los métodos para la clasificación fueron implementados para *Matlab*, una de las características principales que se busca en el proceso de generación de grafos es la capacidad de guardar la información de una manera sencilla y compacta pero que retenga todas las características propias de dicho grafo de tal manera que esta información sea transportable y almacenable para su uso posterior.

Para esto, se creó una estructura de datos la cual contiene, de una manera estandar, toda la información con respecto a los vértices y aristas así como sus respectivas etiquetas. Se optó por usar un archivo *csv*[1] para cada uno de los objetos por su facilidad de lectura por varios programas para procesarlos y manipularlos. Esta estructura consta de 3 partes

[1] **csv**: archivo de texto de valores separados por comas

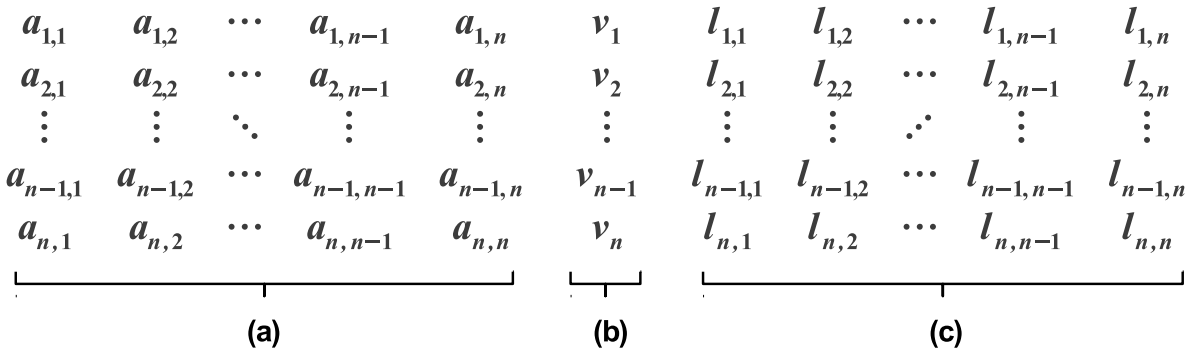


Figura 67. Estructura de datos utilizada para almacenar los grafos. (a) Matriz de adyacencias. (b) Vector de etiquetas de los vértices. (c) Matriz de etiquetas en las aristas.

(ver Figura 67): la matriz de adyacencias (a), el vector con las etiquetas de cada uno de los vértices (b) y la matriz de adyacencias con las etiquetas correspondientes a cada una de las aristas (c).

B.1.2 Topologías

Existen distintas topologías de grafos que se pueden generar con el paquete *igraph*. En este trabajo consideramos las siguientes:

Árbol Requiere tres parámetros: el número de vértices, el número de hijos por vértice y el *modo*, que puede ser "out", "in" o "undirected" en donde los dos primeros indican la dirección en la que apuntan las aristas en caso de ser un árbol dirigido y el último indica que se desea crear un grafo *no dirigido*.

Estrella Requiere tres parámetros: el número de vértices, el *modo* que indica la dirección de las aristas en caso de ser un grafo dirigido y el índice del vértice que será el centro de la *estrella* a crear.

Anillo Requiere cuatro parámetros: el número de vértices, "dirigido" (falso o verdadero), "mutuo" (falso o verdadero) que indica si las conecciones entre vértices serán mutuas en caso de ser un grafo no dirigido y "circular" (falso o verdadero) que indica si el grafo será circular o no.

Red de mundo pequeño Requiere cuatro parámetros: *dimensión* del enrejado[2] inicial, el *tamaño* del enrejado en cada una de las dimensiones, el vecindario dentro del cual los vértices del enrejado serán conectados y la probabilidad de re asignación de aristas. El número de vértices en total es igual a $n = tamaño^{dimensión}$.

Aleatorio Se genera por medio del juego *Erdos Renyi* y requiere cinco parámetros: el número de vértices, p o m que indica la probabilidad p de que exista una arista entre un par de vértices o el número total m de aristas a crear, "tipo" ("gnp" en caso de haber proporcionado la probabilidad o "gnm" en caso de haber proporsionado el número exacto de aristas que se quieren), "dirigido" (falso o verdadero) que indica el modo del grafo y "autociclos" (falso o verdadero) que determina si se permiten aristas al mismo vértice.

B.1.3 Grafos compuestos

Como parte de los casos de pruebas sintéticos se generaron grafos con topologías no convencionales de una manera controlada para así diseñar experimentos más completos y con esto probar más a fondo el desempeño de cada uno de los kernels para grafos.

Con esto en mente, se introducen los *grafos compuestos* que consisten en un conjunto de grafos de topologías conocidas conectados entre si de tal manera que forman una topología "externa". Se pueden ver a los *grafos compuestos* como grafos normales en donde cada uno de sus vértices en realidad es un grafo con topología propia.

B.2 Descripción de métodos para generar grafos

Ya que el paquete *igraph* nos da mucha flexibilidad y variedad en los tipos de grafos que se pueden crear, es necesario implementar programas que permitan generar grafos de manera controlada.

[2]el algoritmo utilizado para generar la *red de mundo pequeño* lo hace a partir de un grafo en forma de *enrejado* o *retícula*.

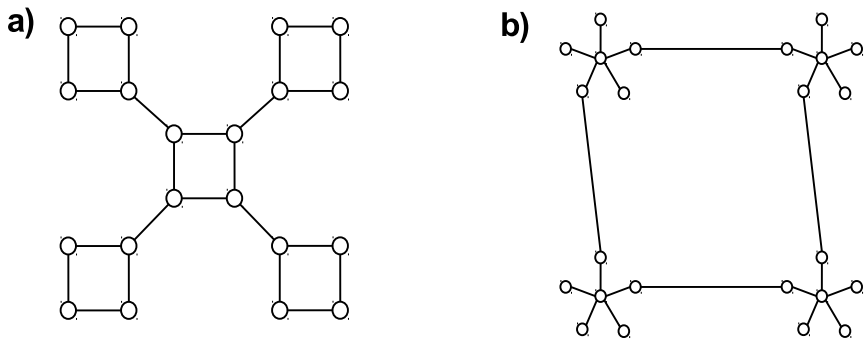


Figura 68. Ejemplos de grafos compuestos. a) Una *estrella* "externa" compuesta de *anillos* "internos". b) Un *anillo* "externo" compuesto de *estrellas* "internas".

Es por esto que se implementaron dos algoritmos para manejar la generación de los casos de prueba. Un algoritmo genera cualquiera de las cinco topologías descritas y el otro genera los grafos compuestos.

B.2.1 Parámetros de métodos para generar grafos

Por la naturaleza distinta de ambos tipos de grafos, es necesario manejar dos algoritmos con parámetros distintos, aunque hay algunos que se repiten para ambos. A continuación se describen los parámetros comunes utilizados por ambos algoritmos así como los que son propios de cada uno:

B.2.1.1 Parámetros comunes

N Es el número de grafos que se quieren generar por cada uno de los *tipos* en el caso de que sean grafos sencillos o de las *formas* en caso de ser grafos compuestos.

Etiquetas de vértices Puede ser dado en forma de un conjunto de etiquetas. Se asigna aleatoriamente una etiqueta dentro de éste a cada uno de los vértices.

Etiquetas de aristas De igual manera que las etiquetas de los vértices, estas pueden venir de un conjunto. También se asigna aleatoriamente una etiqueta del conjunto por cada

arista o se le asigna la misma a todas en caso de existir solo una.

Propósito Para la clasificación es necesario generar un conjunto de grafos para el entrenamiento y otro conjunto para la prueba del algoritmo para determinar la exactitud en la clasificación. Esta opción permite seleccionar si los grafos serán con el propósito de entrenamiento, para pruebas o para ambos.

Este parámetro recibe una lista en la cual pueden estar "entrenamiento" y "prueba" o ambos. En caso de que estén ambos, el algoritmo genera dos conjuntos de pruebas distintos pero generados con las mismas propiedades, uno para cada propósito.

Folder Es el nombre de la carpeta en la que se quiere almacenar toda la información. Por lo regular, se le asignan números para facilitar su manejo.

B.2.1.2 Grafos sencillos

El algoritmo generador de grafos sencillos es capaz de generar cualquiera de las cinco topogías descritas. Debido a la naturaleza de estas cinco topologías distintas, es necesario establecer los parámetros que sean suficientes para generar cualquiera de ellas.

Estos son los parámetros particulares del generador de grafos sencillos:

tipos Es el tipo de grafo que se quiere generar. Puede ser uno solo o también puede contener una lista de tipos de grafos. En el caso de ser una lista, el método generará N grafos por cada elemento de esta lista. Los tipos permitidos son: árbol, estrella, anillo, red de mundo pequeño y aleatorio.

n es el número de vértices de cada grafo. Este parámetro proviene de un conjunto de valores de entre los cuales se selecciona uno aleatoriamente cada vez que se genera uno, de tal manera que se puede tener un conjunto de grafos de tamaños distintos o un conjunto de grafos del mismo tamaño.

p Como parte de los parámetros necesarios para generar un grafo aleatorio, es necesario incluir la probabilidad de que exista una arista entre cada par de vértices del grafo. En caso de que éste sea 0, se asignará una probabilidad aleatoria.

Al igual que el método para generar grafos aleatorios, también se requiere una probabilidad p en el caso de la generación de grafos tipo *red de mundo pequeño*, que es la probabilidad de "re-alambrado".

En el caso de este parámetro común, se recibe un arreglo de dos valores en donde el primero es la probabilidad para generar grafos aleatorios y el segundo es para los grafos de tipo *red de mundo pequeño*.

B.2.1.3 Grafos Compuestos

El algoritmo generador de grafos compuestos es capaz de generar un conjunto de grafos que serán conectados de tal manera que se forme una topología entre ellos. Estas topologías pueden ser: estrella, anillo y árbol.

Para la facilidad de explicación y manejo, se le llama "forma" a la topología en la cual serán conectados el conjunto de grafos a los cuales se les llama "grafos internos".

Los siguientes son los parámetros particulares del generador de grafos compuestos:

forma es la topología en la que se quiere conectar el conjunto de grafos internos que van a formar cada grafo compuesto creado. Esta topología o *forma* puede ser: anillo, estrella o árbol.

número de grafos internos es el número de grafos internos que se quieren conectar para formar una topología. Se puede ver como el número de "vértices" que tendrá el "grafo externo" o forma.

número de vértices en grafos internos es el número de vértices que tendrá cada grafo interno. Este parámetro puede un solo número o un conjunto de valores de los cuales se asigna aleatoriamente uno para cada grafo.

tipos de grafos internos con este parámetro podemos elegir qué topologías tendrá cada uno de los grafos. Este parámetro debe de ser una lista con los tipos deseados o un solo elemento describiendo la topología que se quiere para todos los *grafos internos* del grafo compuesto.

En el caso de que el número de elementos de dicha lista sea más de 1 pero menos que el número total de grafos internos, se recorren uno por uno los elementos de la lista de manera circular. Es decir, cuando llega al último elemento, el siguiente será el primero de nueva cuenta.

B.2.2 Tipos de clases

Dentro del proceso de generación y lectura de los grafos para la realización de los experimentos, dependiendo del tipo de grafo a generar, se puede seleccionar el tipo de "etiqueta" que se le dará a cada objeto por el cual se va a clasificar. Estas etiquetas pueden ser asignadas en el momento en el cual se genera el objeto o más adelante, durante el proceso de lectura del mismo para ser procesado. Todo depende del tipo de experimento que se quiera realizar.

Aparte de la clasificación de enzimas con el conjunto de datos reales con el que se cuenta, existen dos tipos distintos de clases que se pueden aplicar ya sea a grafos sencillos como a grafos compuestos:

Manual Este tipo de etiquetado se asigna en el momento mismo en el que se genera el grafo con el paqueta *igraph*. En este caso, usualmente se quiere clasificar entre dos tipos de grafos, siendo la clasificación por topología lo más sencillo.

Se le asigna un número distinto a cada una de las clases de acuerdo al tipo de grafo y se agrega este número al final del nombre del archivo que contiene la información de cada uno de los grafo generados. De este modo, al momento de ser capturados los archivos correspondientes a cada uno de los grafos, se le asigna una etiqueta a cada uno de ellos dependiendo del dígito que se grabó en su nombre.

Recorridos Este tipo de etiqueteado se realiza por lo regular a un conjunto de grafos del mismo tipo ya que, lo que determinará la clase a la que cada uno de estos grafos pertenecerá es el tipo de recorrido que se les haga. Este proceso es independiente al generado inicial de los grafos, es decir, primero se generan los grafos del mismo tipo y después, como parte del proceso de lectura de los archivos, se realiza el nuevo etiquetado.

Primeramente se obtiene un árbol de esparcimiento mínimo (Cormen *et al.*, 2001) a cada uno de los grafos. Depués, a cada uno de estos árboles de esparcimiento mínimo se les realiza un recorrido, pre-orden (Cairo, 2002) a la mitad y post-orden (Cairo, 2002) a la otra mitad. De acuerdo al recorrido, se etiqueta cada uno de los vértices de manera ascendente. Las etiquetas en este caso pertenecen a un conjunto el cual se puede variar. Por lo regular este conjunto va del 1 al 5 pero puede ser variado fácilmente.

B.2.3 Variación en el número de etiquetas

Además de los experimentos ya citados, también se puede variar el número de etiquetas distintas tanto en vértices como en aristas para evaluar el impacto que tiene en la exactitud de la clasificación.

Esta capacidad fue utilizada más que nada en la etapa de experimentación ya que se decidió utilizar un número de etiquetas fijo, tanto para aristas como para vértices.

B.3 Conjuntos de Datos reales

En esta sección se describen los conjuntos de datos reales que se usarán para la comparación de los métodos kernel tratados en este trabajo. Estos conjuntos de datos son compuestos químicos que pueden ser representados como grafos.

En el caso de los conjuntos *Mutag* (Debnath *et al.*, 1991) y *PTC* (Helma *et al.*, 2001) las etiquetas de los vértices representan los nombres de los átomos y las etiquetas de las

aristas representan el tipo de enlace.

En el caso del conjunto de *Pseudocentros* (Gramada y PE, 2006), los vértices representan pseudocentros asociados a enlaces químicos y las etiquetas de las aristas representan la distancia en armstrongs entre cada pseudocentro.

B.3.1 Mutag

Contiene 230 compuestos químicos (nitrocompuestos aromáticos y heteroaromáticos) en donde el propósito es predecir mutagenicidad en un subconjunto de 188 compuestos considerados apropiados para la clasificación por Debnath *et al.* (1991).

Los 188 compuestos son divididos en 2 grupos: 125 muestras contienen niveles positivos de mutagenicidad y 63 contienen niveles negativos.

B.3.2 PTC

El conjunto de datos PTC (Helma *et al.*, 2001) se obtuvo al suministrar 417 compuestos químicos a cuatro tipos de animales de prueba: ratón macho, ratón hembra, rata macho y rata hembra. Estos animales se identifican de acuerdo a sus nombres en inglés como: MM (male mouse), FM (female mouse), MR (male rat) y FR (female rat), respectivamente.

De acuerdo a su carcinogenicidad, se le asigna una de las siguientes etiquetas a cada compuesto: $\{EE, IS, E, CE, SE, P, NE, N\}$ en donde CE, SE y P indican "relativamente activo", NE y N indican "relativamente inactivo" y EE, IS y E indican "no puede determinarse".

Para simplificar el problema, se les etiqueta como "positivo" a CE, SE y P y como "negativo" a NE, N. El propósito es predecir si cada compuesto es es positivo o negativo para cada tipo de animal de prueba.

B.3.3 Pseudocentros

El conjunto de datos *Pseudocentros* (Gramada y PE, 2006) contiene 40 enzimas en las cuales los amonoácidos son sustituidos por cinco tipos de pseudocentros asociados a enlaces

químicos: Donador, Aceptor, Donador-Aceptor, Alifático y Aromático (correspondiente a las etiquetas DO, AC, DA, AL y PI respectivamente). La posición de estos está dada como el centro geométrico de los átomos responsables de la propiedad que representan (Shulman-Peleg *et al.*, 2004).

Las 40 enzimas son divididas en 2 grupos: 20 proteinas de la familia de las quinasas (EC 2.7.1.-) utilizados por Gramada y PE (2006) y 20 proteinas de las cuales 9 pertenecen al grupo de las tranferasas (EC 2.5.1.-) y 11 al grupo de las "liasas carbón-carbón" (EC 4.1.-.-).

En la Tabla IX se muestra un resumen de algunos estadísticas asociadas a cada uno de los 3 conjuntos de datos utilizados.

Tabla IX. Varios estadísticos de los conjuntos de datos reales.

	MM	FM	MR	FM	Mutag	pseudocentros
# Ejemplares	336	349	344	351	188	40
# Positivos	129	143	152	121	125	20
# Negativos	207	206	192	230	63	20
n Max	109	109	109	109	40	92
n Promedio	25.0	25.2	25.6	26.1	31.4	49.5
$\mid V \mid$	21	19	19	20	8	5
$\mid E \mid$	4	4	4	4	4	10

donde **# Positivos**: ejemplos positivos, **# Negativos**: ejemplos negativos, **n Max**: tamaño máximo de los grafos, **n Promedio**: tamaño promedio de los grafos, $\mid V \mid$: número de etiquetas de vértices, $\mid E \mid$: número de etiquetas de aristas.

B.4 Conjuntos de datos sintéticos

Además de los conjuntos de datos reales, se crearon varios conjuntos de datos sintéticos diseñados para analizar el desempeño de cada uno de los métodos kernel tratados con respecto a cierto parámetro específico.

Esto se realizó fijando los parámetros a utilizar y modificando únicamente el parámetro del cual se quiere determinar su efecto en el desempeño. Con esto se crearon tres grupos de grafos para ser utilizados en distintas pruebas con los métodos kernel implementados y

sus variaciones.

El propósito de estos 3 grupos de grafos es evaluar el impacto de sus respectivos parámetros en el tiempo de cálculo de los kernels. Es por esto que en este caso todos los grafos son aleatorios, es decir, no existe ningún tipo de agrupamiento o distinción por clases en estos conjuntos de datos.

A continuación se describen los distintos grupos de datos sintéticos que se crearon:

B.4.1 Densidad (c)

Se crearon 8 conjuntos de grafos separados de acuerdo a su densidad (c), es decir, la cantidad de aristas. El grupo de datos consta de 8 conjuntos de 64 grafos de 32 vértices cada uno. Cada conjunto varía su densidad que va desde 20% hasta 90%.

Debido a los requerimientos de los métodos kernel y, en algunos de los casos, el tamaño de los grafos a trabajar, no se incluyeron conjuntos de datos de 10% ni 100% de densidad ya que, para el primero, se generarían grafos no conectados y en el segundo caso, los grafos incluirían *autociclos* lo cual no es permitido.

En la Tabla X se muestran los parámetros con los que se generaron los 8 grupos de grafos de acuerdo a su densidad.

Tabla X. Parámetros de grafos del grupo de datos *Densidad*.

Número de grafos por grupo	64
Número de vértices por grafo	32
Densidad	**20% - 90%***
Etiquetas en vértices	$\{1-20\}$
Etiquetas en aristas	$\{1-20\}$

*La densidad de los grafos varía de 20% hasta 90% en incrementos de 10%.

B.4.2 Número de grafos (N)

Se crearon 8 conjuntos de grafos separados de acuerdo a una progresión exponencial del número de grafos en el conjuntos.

El número grafos de cada uno de los conjuntos va desde 8 duplicando su tamaño en cada conjunto hasta llegar a 1024 grafos en el último. El resto de los pámetros (densidad, tamaño, etc) se mantienen fijos. Estos parámetros se muestran en la Tabla XI.

Tabla XI. Parámetros de grafos del grupo de datos *Número*.

Número de grafos por grupo	**8 - 1024***
Número de vértices por grafo	16
Densidad	50%
Etiquetas en vértices	$\{1-20\}$
Etiquetas en aristas	$\{1-20\}$

*El número de grafos en cada conjunto varía de 8 hasta 1024 duplicando su valor entre cada conjunto distinto.

B.4.3 Tamaño de grafos (n)

Se crearon 8 conjuntos de grafos separados de acuerdo a una progresión exponencial del tamaño, en número de vértices, de cada uno de sus grafos.

El tamaño de cada uno de los grafos va desde 8 vértices duplicando su tamaño en cada conjunto hasta 1024 vértices en el último grupo manteniendo en cada uno de los grupos el resto de sus parámetros fijos. Estos parámetros se muestran el la Tabla XII.

Tabla XII. Parámetros de grafos del grupo de datos *Tamaño*.

Número de grafos por grupo	64
Número de vértices por grafo	**8 - 1024***
Densidad	50%
Etiquetas en vértices	$\{1-20\}$
Etiquetas en aristas	$\{1-20\}$

* El tamaño de los grafos (número de vértices) varía de 8 hasta 1024 duplicando su valor entre cada conjunto distinto.

B.4.4 Esquemas de etiquetado

Por lo general, se clasifican objetos que obedecen a cierto patrón en su topología. En este caso, se creó un conjunto de datos el cual consiste en el mismo tipo de grafos separados

únicamente por las etiquetas de los vértices las cuales fueron determinadas de acuerdo a dos *esquemas de etiquetado*.

Estos dos esquemas consisten en obtener un árbol de esparcimiento mínimo (MST) y después recorrer dicho árbol de una de dos maneras: *preorden* y *postorden*, y etiquetar los vértices conforme se vayan pasando por cada uno.

Para esto, se crearon tres grupos en los cuales solo se modifica el número de grafos (N) y el tamaño de cada uno (n) para así determinar, el impacto que tienen estos parámetros sobre la clasificación.

Esquemas #1: N grande y n moderado. La idea es tener el mayor número de grafos posible pero considerando no excederse en el tamaño de cada uno de ellos ya que su análisis resultaría muy costoso. Con esto se intenta evaluar el impacto que tiene el número de grafos a procesar en el tiempo de procesamiento y exactitud en la clasificación.

Esquemas #2: n grande y N moderado. En este caso se generan grafos lo más grande posible sin descuidar el tamaño y buscando un equilibrio entre el mayor tamaño y un número manejable de grafos generados, es decir, el número y tamaño de grafos . De esta manera se busca evaluar el impacto que tiene el tamaño de dichos grafos.

Esquemas #3: N y n moderados. Para propósitos de comparación, se genera un número moderado de grafos de tamaño moderado. En este conjunto de experimentos se espera tener un número N menor de grafos que en el Experimento 2 pero mayor que en el Experimento 3 y por otro lado grafos más grandes a los del Experimento 2 pero más pequeños que los del Experimento 3. Esto nos dará puntos de comparación y nos permitirá ver el impacto que tiene el aumentar el tamaño de los grafos por un lado y el número de estos por el otro.

Tabla XIII. Parámetros de grafos de los tres grupos de datos creados para clasificar por esquemas.

Grupo:	# 1	# 2	# 3
Número de grafos por grupo	600	60	120
Número de vértices por grafo	20-30	60-200	25-60
Densidad	-	-	-
Etiquetas en vértices	$\{1-5\}$	$\{1-5\}$	$\{1-5\}$
Etiquetas en aristas	$\{1-5\}$	$\{1-5\}$	$\{1-5\}$

B.5 Detalles de experimentos

Los experimentos fueron realizados en una máquina con procesador *Intel Core 2 Quad Q9650* a 3.00GHz y 10GB de memoria RAM con sistema operativo Ubuntu Linux 10.10 de 64bits. Los casos de prueba se generaron con el paquete *igraph* para *R*, los kernels para grafos así como los algoritmos de prueba y recolección de datos se programaron en Matlab R2009a.

Para la clasificación se utilizó el paquete *libSVM* (Chang y Lin, 2011) que es ampliamente utilizado en la comunidad científica por su calidad y la disponibilidad de código en una variedad de lenguajes de programación (en este caso se utilizó la versión para Matlab).

La validación cruzada se emplea sobre todo cuando no es posible tener o generar elementos ilimitados para su clasificación. En este caso se selecciona un número k que es el número de grupos que se forman con los objetos a clasificar disponibles y posteriormente se entrena con $k-1$ objetos y se prueba con el restante. Esto se repite con cada uno de los grupos, es decir, k veces.

Para los conjuntos de datos reales, se utiliza $k = n$ a lo que se le llama validación cruzada *LOO* (*Leav One Out* por sus siglas en inglés), esto es, se entrena con todos menos 1 elemento de los disponibles y se verifica con dicho elemento. Puesto que este procesos se repite k veces, solo es posible realizar experimentos utilizando *LOO* con conjuntos de datos relativamente pequeños. En nuestro caso, sí fue posible realizar todos los experimentos con datos reales utilizando esta técnica ya que el conjunto de datos con más elementos es

PTC-FM con 351.

Ya que el desempeño de los dos kernels comparados en este trabajo son dependientes tanto de su implementación como de la interpretación y manejo de los datos de entrada, es injusto comparar los resultados obtenidos en este trabajo con los obtenidos en sus respectivos documentos.

Por tal motivo, se comparan los resultados obtenidos con nuestra implementación y forma de tratar los conjuntos de datos de los kernels principales, es decir, el *kernel de caminos más cortos* y el *kernel de sub-árboles* y se evalúa el impacto de las variaciones con respecto a su desempeño.

Para propósitos de visualisación y mejor manejo del tamaño de los nombres, le llamaremos al kernel de *caminos más cortos SP* (por sus siglas en inglés "Shortest Path") y al kernel de *subárboles Weisfeler-Lehman WL* y se le agregará su versión correspondiente al kernel básico utilizado. Por ejemplo, al kernel de *caminos más cortos* que emplea el kernel *lineal* (ecuación 1) se le llamará *SP-lineal*.