PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

ESCUELA DE INGENIERIA

ALGORITMO DE OPTIMIZACIÓN PROBABILÍSTICO PARA RESOLVER EL PROBLEMA DE SELECCIÓN DE MODELO

ESTEBAN CORTAZAR MORIZON

Tesis para optar al grado de

Magister en Ciencias de la Ingeniería

Profesor Supervisor:

DOMINGO MERY

Santiago de Chile, Enero, 2012

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

ESCUELA DE INGENIERIA

ALGORITMO DE OPTIMIZACIÓN PROBABILÍSTICO PARA RESOLVER EL PROBLEMA DE SELECCIÓN DE MODELO

ESTEBAN CORTAZAR MORIZON

Tesis presentada a la Comisión integrada por los profesores:

DOMINGO MERY

JORGE BAIER

GONZALO ACUÑA

CHRISTIAN OBERLI

Para completar las exigencias del grado de

Magister en Ciencias de la Ingeniería

Santiago de Chile, Enero, 2012

A mi padre, por su enorme ayuda a lo largo de todos mis estudios.

AGRADECIMIENTOS

Me gustaría agradecer a todos los miembros de GRIMA, alumnos y profesores, por su ayuda y apoyo a lo largo de esta investigación. Específicamente, agradezco a Álvaro Soto, Karim Pichara y Jorge Baier por discusiones muy beneficiosas para este trabajo. También me gustaría agradecer a mi profesor supervisor, Domingo Mery, por su confianza en mí y constante guía durante este trabajo. Por último, hago un gran agradecimiento a mi familia y amigos. En especial, a mis padres por incentivarme a hacer esta investigación y por su enorme apoyo y ayuda a lo largo de ella.

Este trabajo fue financiado, en parte, por fondos del Fondecyt-Chile, número 1100830, titulado "*Automated X-ray testing of complex structures using efficient view search, view planning and active learning*".

INDICE GENERAL

Pág.

INDICE DE TABLAS

Pág.

INDICE DE FIGURAS

Pág.

RESUMEN

En aprendizaje supervisado existen numerosos algoritmos de preprocesamiento, selección de variables y clasificación, cada uno de los cuales tiene parámetros que permiten ajustarlo. El conjunto de combinaciones de algoritmos y parámetros (modelos) es enorme. Además, cada posible modelo debe ser entrenado y probado numerosas veces para una evaluación certera de su poder predictivo. Esta combinación de espacio de búsqueda grande y tiempo de evaluación elevado hace que encontrar un buen modelo puede tomar mucho tiempo. Por ejemplo, en el capítulo de experimentos, se puede ver que PSMS (un algoritmo de estado del arte, específicamente diseñado para la selección de modelo) requiere de más de dos semanas y media en la búsqueda de un modelo para un conjunto determinado de datos.

Esta investigación tiene como objetivo diseñar e implementar un algoritmo que logre encontrar el mejor modelo posible en un tiempo razonable. Para esto, se basa inicialmente en un algoritmo ya validado, llamado paramILS. Este, propuesto por Hutter et al (2009), es un algoritmo de optimización diseñado para calibrar parámetros de modo de disminuir el tiempo de ejecución de programas complejos. Luego, manteniendo la misma trayectoria de búsqueda de paramILS, se propone en esta tesis un nuevo algoritmo (PILS), el cual incorpora técnicas probabilísticas de modo de poder trabajar con una función objetivo estimada. Esta función objetivo retorna un estimador ruidoso de la calidad de cada modelo candidato, a cambio de una ganancia sustancial en tiempo de ejecución.

Los resultados mostraron que no solo se logró resolver el problema con un desempeño comparable al de algoritmos del estado del arte en este contexto, sino que se hizo utilizando entre el 5% y el 20% del tiempo requerido por los mismos.

Palabras Claves: Selección de modelo, Optimización Probabilística, ILS, paramILS, PSO, PSMS

ABSTRACT

In supervised learning several preprocessing, feature selection and classification algorithms exist. Each one of these algorithms has several parameters that enable tuning. The resulting set of possible combinations of algorithms and parameters (models) is huge. Moreover, in order to evaluate the quality of a candidate model with high certainty, several training and testing procedures must be executed. This combination of large search space and long evaluation time makes searching for a model a long task. As an example, in the experiments chapter of this thesis, PSMS (a state of the art algorithm, specifically designed for model selection) requires more than two and a half weeks to find a model for a certain dataset.

This research attempts to design and implement an algorithm that is capable of finding the best possible model in a reasonable time. In order to achieve this, an initial proposal was based on a validated algorithm, called paramILS. This algorithm, proposed by Hutter et al (2009), is an optimization algorithm designed for parameter tuning for diminishing runtime of complex programs. Then, maintaining the search trajectory of paramILS, a new optimization algorithm is proposed (PILS), which incorporates probabilistic techniques in order to work with an estimation of the objective function. This new objective function returns a noisy estimator of the quality of each candidate model, in exchange of a substantial decrease in runtime.

The results showed that not only was the problem solved with a comparable performance to state of the art algorithms in this context, but that it was achieved using between 5% and 20% of the time required by them.

Keywords: Full Model Selection (FMS), Probabilistic Optimization, Iterative Local Search (ILS), paramILS, Particle Swarm Optimization (PSO), Particle Swarm Model Selection (PSMS)

1. INTRODUCCIÓN

En Ciencias de la Computación, específicamente en el área de Inteligencia Artificial, el problema de aprendizaje supervisado es de enorme relevancia. Este se define como: a partir de un conjunto de datos, donde cada muestra tiene explicitada su clase, buscar patrones con el objetivo de predecir la clase de nuevas muestras. Para lograrlo, la lista de algoritmos de preprocesamiento, selección de variables y clasificación es muy grande y, debido a la importancia del problema, se encuentra en constante crecimiento. Para un análisis más detallado del tema y posibles algoritmos disponibles ver, por ejemplo, Bishop (2006).

Este trabajo se encuentra enfocado a la resolución del problema de selección de modelo o *Full Model Selection* (FMS). Este término puede ser usado en diversos contextos, pero en este trabajo se utilizó la definición de Escalante et al (2009) la cual dice: dado un conjunto de datos específico; una batería de algoritmos de preprocesamiento, selección de variables y clasificación y los parámetros utilizados para calibrarlos, encontrar la mejor combinación de algoritmos y parámetros (de ahora en adelante modelo) para ese conjunto de datos (ver **Figura 1)**.

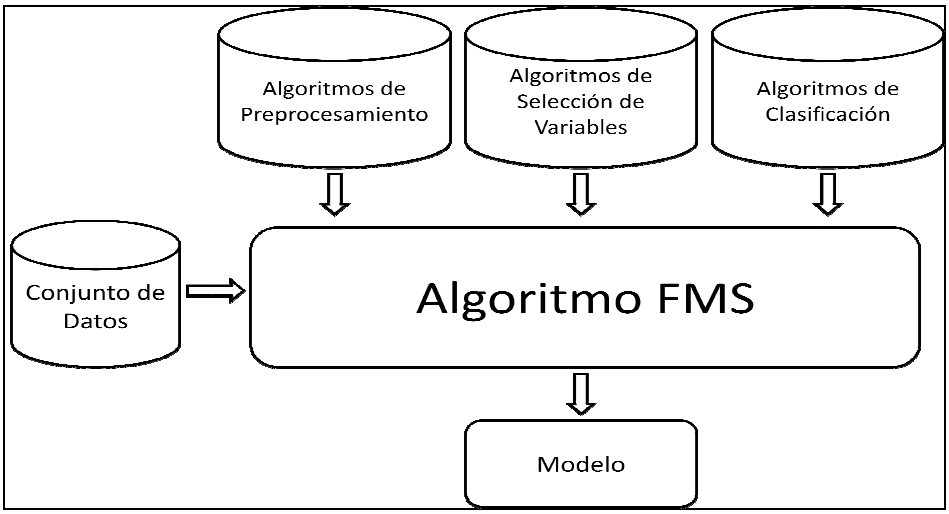

Figura 1: Diagrama del problema de selección de modelo o *Full Model Selection* (FMS).

Resulta importante recalcar que existen numerosos algoritmos de preprocesamiento, de selección de variables y de clasificación, y que cada uno de estos tiene parámetros que pueden ser ajustados para mejorar su desempeño. Considerando que cada modelo es una combinación de estos algoritmos y parámetros, el espacio de búsqueda de posibles modelos es enorme. Además, la evaluación de un modelo es una tarea que puede tomar mucho tiempo. Esto se debe a que para conseguir una estimación de la calidad de un modelo, con un alto nivel de certidumbre, este debe ser entrenado y probado muchas veces. La combinación de un espacio de búsqueda grande con un tiempo de evaluación largo, resulta en una tarea de optimización que puede tomar mucho tiempo. A modo de ejemplo, en el capítulo de experimentos, se puede ver que PSMS (un algoritmo de estado del arte, específicamente diseñado para la selección de modelo) requiere de más de dos semanas y media en la búsqueda de un modelo para un conjunto determinado de datos. El tiempo requerido hace que esta tarea sea inviable para un investigador.

A primera vista se podría asumir que este es un problema que se puede resolver de modo a priori, sin la necesidad del conjunto de datos específico, basándose

únicamente en la calidad de los diferentes modelos, pero no es el caso. El motivo por el cual no es posible está demostrado por el llamado *No Free Lunch Theorem* postulado por Wolpert et al (1997). Este teorema dice que en un problema de optimización, al alejarse de un modelo general es posible mejorar el desempeño para una clase de problemas, pero que eso empeorará, necesariamente, el desempeño para otras clases de problemas. En base a este teorema resulta claro que se requiere de un modelo específico al problema, en lugar de uno general que pueda ser definido a priori. Este argumento ha sido estudiado en profundidad y existen numerosas publicaciones que muestran las ventajas de un enfoque menos generalista en el proceso de selección de modelo o calibración de parámetros, como por ejemplo el trabajo presentado por Smit et al. (2010).

En la actualidad, investigadores realizan el proceso de selección de modelo utilizando convenciones (por ejemplo, el número de vecinos utilizados para KNN debería ser relativamente bajo), decisiones ad hoc (por ejemplo, SVM ha funcionado bien antes, probablemente funcione bien ahora) y haciendo experimentación sobre un subconjunto muy acotado de posibles modelos (por ejemplo, comparar el desempeño de tres clasificadores con sus parámetros por defecto). El problema con este enfoque es que, aunque retorna resultados aceptables, se limita a modelos generalistas. Esto, como fue mencionado en el párrafo anterior, permite tener buenos resultados en la mayoría de los casos pero pierde las ventajas que podrían obtenerse de modelos más específicos.

Otro enfoque para la búsqueda de modelos específicos por sobre modelos generalistas, ha sido la construcción de ensambles de clasificadores, ver Polikar (2006). Este enfoque parte de la premisa de que es posible construir un modelo específico, mediante la unión de múltiples modelos generalistas. Esta metodología ha demostrado ser muy efectiva, pero agrega complejidad al modelo resultante. En mi investigación, se tiene como objetivo la búsqueda de un solo modelo, altamente específico, que resuelva el problema de aprendizaje supervisado.

Existen numerosos algoritmos de optimización ampliamente discutidos en la literatura. Entre ellos, un modelo muy simple con una estructura similar a la de un algoritmo genético es el de *Iterative Local Search* (ILS). No es posible definir ILS como un algoritmo de optimización en sí, sino que debe ser catalogado como una meta-heurística. Esto se debe a que ILS requiere de la definición de un número de funciones para ser implementado. Una implementación de ILS es paramILS, presentado por Hutter et al (2009). La construcción de paramILS utiliza la estructura ILS, para luego definir además la trayectoria de búsqueda y el modo de evaluación de la función objetivo. El objetivo de este algoritmo es la selección de parámetros para la optimización del tiempo de ejecución de algoritmos. Resulta fácil ver cómo esta definición particular de un ILS puede ser útil para problemas como el propuesto en este trabajo.

Como fue mencionado anteriormente, la correcta evaluación de cada modelo candidato puede requerir de mucho tiempo. Esto se debe a que la evaluación de un modelo intenta estimar el poder predictivo del mismo. Para obtenerlo, primero se debe entrenar el modelo sobre el conjunto de datos, proceso en el que se buscan los patrones que permiten definir la clase de cada muestra. Luego, se debe evaluar este modelo sobre un conjunto de datos diferente y medir su desempeño. La metodología comúnmente usada, consiste en dividir el conjunto de datos inicial en uno de entrenamiento y uno de prueba. El problema que conlleva esta metodología es que puede haber ruido en la evaluación, causado por una mala división de los datos. Para solucionar este problema, generalmente se repite el proceso varias veces para asegurar una correcta evaluación del poder predictivo del modelo. Es esta continua división, entrenamiento y evaluación la que hace que el proceso de evaluación de un modelo tome mucho tiempo.

Aunque no existe mucha investigación en la resolución del problema de selección de modelo, éste no es nuevo. Un buen referente para esta área es el *Performance Prediction Challenge*, organizado por Guyon et al (2006). Este evento provee un conjunto de datos y una batería de algoritmos, lo cual permite una comparación justa entre sistemas de selección de modelo. Un modelo que demostró ser

tremendamente efectivo en su búsqueda fue el presentado por Escalante et al (2009). Este algoritmo *Particle Swarm Model Selection* (PSMS), basado en *Particle Swarm Optimization* (PSO), obtiene resultados muy buenos en tiempos razonables. A lo largo de mi trabajo se utilizan los mismos conjuntos de datos y algoritmos que se proveen para este desafío, de modo de poder usarlo como punto de referencia. El objetivo es comparar, tanto el desempeño del modelo obtenido como el tiempo necesario para encontrarlo, con PSMS.

El objetivo de este trabajo es solucionar el problema de selección de modelo, siendo de principal importancia la reducción del tiempo de búsqueda necesario. Como fue mencionado en los párrafos anteriores, existen dos motivos por los que la búsqueda de un modelo toma mucho tiempo: Primero, el tamaño del espacio de búsqueda es muy extenso. Segundo, el tiempo necesario para la correcta evaluación de cada modelo es elevado. En esta investigación se atacan ambos problemas.

La hipótesis es: Utilizando como base un algoritmo de optimización ya validado, es posible construir un nuevo algoritmo que reduzca el tiempo de búsqueda gracias a la utilización de una técnica probabilística. El objetivo es rescatar la trayectoria de búsqueda del algoritmo de optimización para la reducción del espacio de búsqueda evaluado y disminuir el tiempo de evaluación de los modelos candidatos a través del uso de un estimador de la función objetivo que sacrifique certidumbre a cambio de tiempo de ejecución.

La metodología formulada para este trabajo es la siguiente:

1. Se utiliza la definición hecha en paramILS, adaptando el algoritmo para resolver el problema de selección de modelo. Este algoritmo ya ha sido validado en la literatura y permite asegurar una trayectoria de búsqueda relativamente eficiente. Así, se reduce enormemente el espacio de búsqueda evaluado en la búsqueda del modelo.

2. Se modifica la propuesta de paramILS para la construcción de un nuevo algoritmo, al que llamamos *Probabilistic Iterative Local Search* (PILS). El enfoque de PILS es rescatar la trayectoria de búsqueda definida por paramILS pero, a través de una técnica probabilística, reducir el tiempo utilizado en evaluaciones de la función objetivo. Esto es posible ya que se trabaja con un estimado de la función objetivo, la que sacrifica certidumbre a cambio de un ahorro en tiempo de ejecución e intenta reducir la incertidumbre únicamente para las soluciones que proyecta podrían ser óptimas.
3. Se validan los resultados obtenidos en base a tres series de pruebas. La primera, es una prueba de concepto sobre datos simulados. Esta tiene por objetivo un análisis del algoritmo propuesto en un ambiente controlado. La segunda, es una comparación entre en algoritmo propuesto y el algoritmo base. Esta tiene por objetivo la validación de las modificaciones hechas al algoritmo base. Finalmente, una comparación entre el algoritmo propuesto y PSMS. Esta tiene por objetivo validar el algoritmo propuesto, comparándolo con uno del estado del arte en este contexto.

Este texto se encuentra organizado del siguiente modo: El Capitulo 1 (Introducción), entrega una mirada general al problema y describe por qué éste es relevante. El Capitulo 2 (Marco Teórico), describe el funcionamiento de los algoritmos utilizados, tanto el algoritmo utilizado como base (paramILS) como el utilizado en la comparación de nuestros resultados (PSMS), y las opciones de función objetivo. El Capitulo 3 (Diseño), explica el enfoque utilizado en la construcción del algoritmo propuesto (PILS) y el funcionamiento del mismo. El Capitulo 4 (Experimentos y Resultados), muestra los experimentos y resultados con los que el algoritmo propuesto fue validado. Finalmente, el Capitulo 5 (Conclusiones) entrega nuestras conclusiones.

2. MARCO TEÓRICO

Esta sección tiene como objetivo explicar el funcionamiento y teoría detrás de los diversos componentes utilizados en esta investigación. Con esto en mente, es posible dividir este capítulo en dos secciones. La primera es la de Algoritmos de Optimización. En este se explicará tanto el algoritmo base utilizado (paramILS), como el algoritmo de selección de modelo con el cual se comparó el desempeño del diseño propuesto (PSMS). La segunda es la de Funciones Objetivo. En éste se plantean las técnicas de evaluación de modelo relevantes en este trabajo.

2.1. Algoritmos de Optimización

El problema de selección de modelo es, en esencia, un problema de optimización. Para esta interpretación es necesaria la definición del espacio de soluciones, una función objetivo y un algoritmo de optimización que recorra el espacio, de acuerdo a algún criterio, para encontrar la solución que tenga la mejor evaluación según la función objetivo. En este punto se describen dos técnicas de optimización que han sido de enorme importancia para este trabajo. La primera es ILS y su implementación enfocada a la calibración de parámetros para disminuir el tiempo de ejecución de algoritmos, paramILS. Esta técnica fue utilizada como base para la construcción del algoritmo propuesto. La segunda es PSO y su aplicación al problema de selección de modelo, PSMS. Este algoritmo intenta resolver el mismo problema propuesto para este trabajo y, por tanto, fue fijado como un punto de referencia para medir el desempeño del algoritmo propuesto.

En paralelo al trabajo de Escalante et al (2009) en PSMS, Gorissen et al (2008) también trabajaba en la construcción de un algoritmo de selección de modelo. Su investigación se enfocó en la aplicación de algoritmos genéticos para resolver el problema de selección de modelos en tareas de regresión. Resulta importante recalcar que su trabajo se realiza principalmente sobre conjuntos de datos de baja dimensionalidad (la mayoría de los resultados reportados son sobre bases de datos de

dos dimensiones). El mayor problema con su propuesta es que requiere la definición de numerosos operadores de mutación distintos. Tanto en PSMS como en el algoritmo propuesto en este trabajo se privilegian las ventajas de técnicas más simples y por ende más adaptables.

2.1.1. Algoritmo Base (ILS/paramILS)

Como fue mencionado anteriormente, en la primera etapa de esta investigación se aplicó un algoritmo de optimización directamente a la resolución del problema de selección de modelo. El algoritmo de optimización utilizado fue paramILS. Este algoritmo es una definición de la meta-heurística llamada ILS. En los siguientes dos puntos se explica el funcionamiento general de un ILS y las definiciones hechas por Hutter et al (2009) para la construcción de paramILS.

Iterative Local Search (ILS)

Iterative Local Search (ILS) es una meta-heurística de optimización muy simple que tiene, únicamente, dos operadores. A continuación se presenta la estructura general de esta meta-heurística en el **Algoritmo 1**.

Algoritmo 1: Iterative Local Search (ILS)

```
Parámetros:
   -  x_inicial: Solución Inicial
Retorna:
      Mejor solución encontrada.

ILS (x_inicial)
{
      x* = BúsquedaLocal(x_inicial);
      loop CriterioTermino ()
      {
            x'  = Perturbación(x*);
            x'' = BúsquedaLocal(x');
            if Mejor(x'',x*)
                  x* = x'';
      }
      return x*;
}
```

Cualquier implementación de esta meta-heurística requiere de la definición de los operadores de `Perturbación` y `BúsquedaLocal`. Aunque para su correcto funcionamiento, es necesaria la definición de la función `Mejor`, este no es, generalmente, un tema de gran interés. Normalmente, ésta se implementa como una comparación directa de la evaluación de una función objetivo sobre cada solución, por lo que se deja la definición de la función objetivo al usuario que utilice el algoritmo, ya que depende del contexto. En el caso del algoritmo propuesto en esta tesis, se hacen modificaciones a la estructura de la función `Mejor`. Es por esto que a continuación se describe una noción general del enfoque que debe tener cada uno de los operadores, además de una breve discusión de las características que deben tener y del principio sobre el cual se sustenta este algoritmo, terminando, finalmente, con una pequeña discusión de la función `Mejor`.

El operador de `Perturbación` modifica una solución candidata para crear una nueva candidata en la vecindad de la original. El modo en el cual hace esto difiere dependiendo de la implementación del algoritmo. Una característica importante de este operador es que debe generar una solución nueva que sea lo suficientemente distinta de la original como para no ser atraído nuevamente al mismo óptimo local, pero no lo suficiente como para empezar nuevamente la búsqueda.

El segundo operador es el de `BúsquedaLocal`. Este operador recibe como parámetro una solución factible para luego buscar un óptimo cercano al mismo. Al igual que el operador anterior, el modo en el cual hace esto difiere dependiendo de la implementación del algoritmo. La ventaja es que este método puede ser rápido en su búsqueda, ya que requiere únicamente encontrar un óptimo local cercano al punto de origen.

La noción sobre la cual se sustenta esta meta-heurística es el *Proximate Optimality Principle*, discutido por Glover et al (1998). Este principio asume que buenas soluciones son similares entre sí. Este supuesto es muy razonable en problemas como el planteado para este trabajo. Basándose en este principio, un ILS salta de un óptimo local a otro cercano de modo aleatorio, quedándose finalmente con el mejor óptimo encontrado.

En la mayoría de los casos, la función `Mejor` depende más del problema al cual se aplica el algoritmo que del diseño del algoritmo de optimización en sí. En otras palabras, en el diseño de un algoritmo de optimización basado en esta meta-heurística, esta función se implementa únicamente como una comparación directa de la calidad dos soluciones candidatas. A lo largo de este trabajo, se construyó un algoritmo que incorpora el modo de evaluación de las soluciones candidatas en su funcionamiento. Gracias a esto, es posible utilizar estimadores de la calidad de cada solución que requieren considerablemente menos tiempo en su evaluación. El motivo por el cual es relevante esta función para el algoritmo propuesto, es que maneja el nivel de

incertidumbre de cada solución candidata, sacrificando tiempo de procesamiento en la disminución de la incertidumbre únicamente para soluciones que parecen ser óptimas.

Parameter Iterative Local Search (paramILS)

Este algoritmo es una implementación particular de la meta-heurística ILS, presentado por Hutter et al (2009). El enfoque para el cual se desarrolló este algoritmo fue la selección y calibración de parámetros que optimicen el tiempo de ejecución de algoritmos de alta complejidad computacional. Resulta fácil ver como este enfoque se asemeja a las necesidades planteadas para este trabajo. Es por eso que se tomó este algoritmo como una base para la construcción del algoritmo propuesto. Para esta implementación particular de ILS, se requiere de la definición de los conceptos claves de la meta-heurística inicial.

El primer concepto y la base sobre la cual está construido el algoritmo es el de vecindad. Este algoritmo define vecindad de una solución, como cualquier solución factible que difiere de la solución inicial en un solo parámetro. En base a este concepto, se crean dos operadores fundamentales para el funcionamiento del algoritmo. El primero es `RandomNbh(x)`, operador que salta a un vecino aleatorio de x. El segundo es `Nbh(x)`, operador que lista en orden aleatorio a todos los vecinos de la solución x.

Por otra parte, cualquier implementación de ILS requiere de la definición de un operador de `Perturbación`. La forma en la cual paramILS construye este operador es muy simple. En primer lugar, se define una constante s. Luego, en cada iteración, se hacen s saltos sucesivos de vecino en vecino desde la solución inicial. La definición de s es una parte importante para el funcionamiento del algoritmo. Considerar un s muy alto hace que el algoritmo se aleje demasiado de la vecindad donde se está buscando el óptimo. Considerar un s demasiado pequeño deja una probabilidad demasiado alta de volver a caer en el mismo óptimo local de la iteración anterior.

El segundo operador importante para la implementación de un ILS es el de `BúsquedaLocal`. Este algoritmo utiliza una técnica llamada *Iterative First Improvement*. Esta consiste en listar todos los vecinos de una solución inicial en orden aleatorio. Luego, se comparan uno a uno contra la solución inicial. En el momento en el que se encuentra una solución mejor que la inicial, se inicia nuevamente el proceso desde esta nueva solución. El código de este método se puede ver en el **Algoritmo 2**.

Algoritmo 2: Iterative First Improvement

```
Parámetros:
   -  x: Solución Inicial desde donde hacer la búsqueda.
Retorna:
      Un óptimo local cercano a la solución inicial entregada.

IterativeFirstImprovement(x)
{
      do
      {
            x' = x;
            for each x'' in Nbh(x)
            {
                  if Mejor(x'', x')
                  {
                        x = x'';
                        break;
                  }
            }
      } while (x' != x);

      return x;
}
```

Una vez definidos los operadores de `BúsquedaLocal` y de `Perturbación`, la implementación de este algoritmo, presentada en el **Algoritmo 3**, es bastante similar a la definida por ILS. La única modificación es la definición del punto de partida del algoritmo. Este algoritmo incorpora un paso adicional en su inicialización, en donde se evalúan r muestras aleatorias y se selecciona la mejor como solución inicial. Aunque

este paso no es completamente necesario para el correcto funcionamiento del algoritmo, considerar un mejor punto de partida disminuye significativamente el número de iteraciones necesarias para encontrar el óptimo. Además, este paso adicional ayuda a que el algoritmo tenga más estabilidad en sus soluciones.

Algoritmo 3: paramILS

```
Parámetros:
   -  x_inicial: Solución Inicial
Constantes:
   -  r: Número de soluciones consideradas como solución inicial.
   -  s: Número de saltos hechos en cada perturbación.
Retorna:
      Mejor solución encontrada.

paramILS(x_inicial)
{
      for i=1..r
      {
                  x = randomX();
                  if Mejor (x, x_inicial)
                        x_inicial = x;
      }

      x* = IterativeFirstImprovement (x_inicial);
      while CriterioTermino()
      {
            x = x*;

            for i=1..s
                  x = RandomNbh(x);
            x = IterativeFirstImprovement (x);

            if Mejor(x, x*)
                  x* = x;
      }

      return x*;
}
```

El algoritmo paramILS tiene dos variantes. La primera es BasicILS, la segunda es FocusedILS. La única diferencia entre ellas es la implementación de la función `Mejor`. En cada caso se maneja la incertidumbre asociada a la medición de la función objetivo de un modo distinto. En el caso de BasicILS, simplemente se resuelve el problema haciendo un número N fijo de evaluaciones de cada solución candidata y luego comparando el promedio. En FocusedILS, se introduce el concepto de dominancia. Este concepto dice que una solución domina a otra únicamente si tiene un mejor promedio, bajo un número mayor o igual de evaluaciones. De este modo se consigue un manejo selectivo del número de evaluaciones. Empíricamente, se observó que las ventajas que daba este manejo dinámico de la incertidumbre de no compensaba el requerimiento adicional de tiempo de cómputo. Es por esto que a lo largo de esta investigación se comparan los resultados utilizando el llamado BasicILS. Para un análisis más detallado de este algoritmo se recomienda el texto, escrito por Hutter et al (2009), al respecto.

2.1.2. Algoritmo Referencia (PSO/PSMS)

Para medir la calidad del algoritmo propuesto en este trabajo, es necesario tener un punto de referencia. Para esto, utilizamos el algoritmo de optimización PSO, específicamente, la aplicación de PSO hecha por Escalante et al (2009) a la resolución del problema de selección de modelo, PSMS. El motivo por el que se seleccionó este algoritmo como referencia fue por su excelente desempeño en la competencia de selección de modelo organizada por Guyon et al (2006).

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) es un algoritmo de búsqueda, basado en poblaciones, inspirado en el comportamiento de comunidades biológicas en que sus miembros tienen metas, tanto individuales como colectivas. Entre estas comunidades pueden listarse bandadas de aves, cardúmenes de peces o enjambres de abejas. Todas estas comunidades tienen en común que sus miembros tienen metas globales, las cuales

son alcanzadas al explorar el medio en el que se encuentran. Este algoritmo fue propuesto por primera vez por Kennedy et al (1995), y desde entonces ha tenido un sinnúmero de aplicaciones en variados escenarios. Su popularidad se debe, en gran medida, a la simpleza del mismo, acoplado con la eficacia de sus resultados. Al igual que los algoritmos genéticos, PSO se encuentra diseñado para manejar espacios de búsqueda muy grandes con muchos óptimos locales, donde un algoritmo de optimización tradicional tomaría demasiado tiempo o sería poco efectivo en su búsqueda.

La estructura de PSO es similar a la de un algoritmo genético, en cuanto a que mantiene una población de soluciones candidatas, las cuales van mutando iteración a iteración. A diferencia de un algoritmo genético, la estructura de PSO se basa en únicamente un operador. Este es el encargado de actualizar a los individuos de la población en cada iteración.

En PSO, cada individuo de la población es llamado una partícula. En cada instante de tiempo t, cada particula tiene una posición en el espacio de búsqueda, como la definida en **(2.1.1)**. Se define un enjambre como un conjunto de partículas, ver **(2.1.2)**. Además, cada partícula tiene asociada una velocidad, que define cuánto se mueven dentro del espacio de búsqueda. La velocidad de cada partícula se define en **(2.1.3)**, donde $v_{i,k}^t$ es la velocidad en la dimensión k de la partícula i en el tiempo t.

$$x_i^t =< x_{i,1}^t, x_{i,2}^t, \dots, x_{i,d}^t > \tag{2.1.1}$$

$$S = \{x_1^t, x_2^t, \dots, x_m^t\} \tag{2.1.2}$$

$$v_i^t =< v_{i,1}^t, v_{i,2}^t, \dots, v_{i,d}^t > \tag{2.1.3}$$

En cada nueva iteración, se actualiza la posición de cada partícula como muestra la ecuación **(2.1.4)**. Además, la velocidad de cada partícula es corregida de acuerdo a la ecuación mostrada en **(2.1.5)**.

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.1.4}$$

$$v_i^{t+1} = W \times v_{i,j}^t + c_1 \times r_1 \times \left(p_{i,j} - x_{i,j}^t\right) + c_2 \times r_2 \times \left(p_{g,j} - x_{i,j}^t\right) \tag{2.1.5}$$

donde $p_{i,j}$ es el valor de la dimensión j de la mejor solución encontrada hasta ahora para la partícula i, solución llamada p_i. También, se considera la mejor solución que ha encontrado el enjambre completo hasta el momento, catalogada de "partícula líder" y denominada p_g. Resulta importante recalcar que, a través de p_i y p_g, el algoritmo toma en cuenta tanto la información individual (local) como la colectiva (global) para la actualización de la velocidad y posición en cada iteración. Tomando esto en consideración, c_1 y c_2 son constantes que ponderan el valor de la mejor solución global y la de la local, r_1 y r_2 son variables aleatorias que distribuyen uniforme entre 0 y 1 (introduciendo una componente azarosa en el proceso). Por último W, es una variable de inercia del sistema. El objetivo de ésta, es controlar el impacto de la velocidad antigua sobre la nueva. El pseudocódigo para PSO se presenta en el **Algoritmo 4**.

Algoritmo 4: Particle Swarm Optimizacion (PSO)

```
Constantes:
      - M: Número de partículas en el enjambre.
      - I: Número de iteraciones.
Funciones Auxiliares:
      - FunciónObjetivo(x): En base a una partícula x, retorna el
        valor de la función objetivo. (Considerando un problema de
        maximización)
      - CrarEnjambreAleatorio(M): Crea M soluciones factibles
        aleatorias.
      - ActualizarVelocidad(...): Retorna la nueva velocidad para una
        partícula de acuerdo a la ecuación (2.1.5)
Retorna:
      Mejor solución encontrada.
PSO ()
{
      S = CrarEnjambreAleatorio(M);
      for j=1..M
      {
            pi(j) = S(j);
            if (FuncionObjetivo(S(j)) > FuncionObjetivo (pg))
                  pg = S(j);
      }
      for i=1..I
      {
            for j=1..M
            {
                  v(j) = ActualizarVelocidad(S(j), v(j), pi(j), pg);
                  S(j) = S(j) + v(j);
                  if ( FuncionObjetivo(S(j)) > FuncionObjetivo(pi(j)) )
                        pi(j) = S(j);
                  if (FuncionObjetivo(S(j))>FuncionObjetivo(pg))
                        pg = S(j);
            }
      }
      return pg;
}
```

Particle Swarm Model Selection (PSMS)

El algoritmo *Particle Swarm Model Selection* (PSMS) es la aplicación, hecha por Escalante et al (2009), de PSO al problema de selección de modelo. Dado que una de las mayores ventajas de PSO es su simpleza, su aplicación a la resolución del problema de selección de modelo es bastante directa. Sin embargo, existe un par de detalles de implementación que merecen ser mencionados.

La modificación más importante al algoritmo planteado en el punto anterior, hecha por Escalante et al (2009), es a la implementación de W. Como fue mencionado, W representa la inercia del sistema. El objetivo de esta inercia es controlar el peso que tiene la velocidad actual en el cálculo de la velocidad nueva en cada iteración. En esta implementación se utilizó una inercia adaptativa, especificada por tres valores **(2.1.6)**, donde, W_{start} y W_{end} son las inercias iniciales y finales respectivamente, y W_f indica la fracción de las iteraciones en la que se disminuye el valor de W. Esta configuración permite la exploración de un gran área del espacio de búsqueda en las primeras iteraciones y luego refinar lentamente la búsqueda en los pasos finales.

$$W = (W_{start}, W_f, W_{end}) \tag{2.1.6}$$

Además de modificaciones a PSO, el trabajo discutido exhibe la representación utilizada por PSMS para el espacio de búsqueda y la forma en la cual se evalúa la calidad de cada solución. Estos elementos son claves, ya que fueron considerados para la construcción del algoritmo propuesto.

En cuanto a la representación del espacio de búsqueda, en PSO cada solución candidata es considerada una partícula. Cada partícula es representada por su posición en el espacio de búsqueda. Esta posición es un vector numérico de d dimensiones. Para el problema de selección de modelo, cada solución candidata es una posible combinación de algoritmos y parámetros, por lo que es necesario codificar las partículas para

representar todos los posibles modelos considerados. En la práctica, cada partícula se representa por un vector que es divisible en las siguientes secciones:

- Los primeros elementos del vector representan los métodos de preprocesamiento. Estos son variables binarias, ya que se permite que un modelo contenga varios métodos de preprocesamiento.
- Los siguientes elementos son los parámetros de cada uno de los algoritmos de preprocesamiento mencionados anteriormente, por separado.
- Luego, una variable entera que representa el algoritmo de selección de variables utilizado. Esto se debe a que sólo es necesario un algoritmo de selección de variables por modelo.
- Los siguientes son los parámetros de los algoritmos de selección de variables mencionados, por separado.
- Luego, una variable entera define qué algoritmo de clasificación se utilizará.
- Finalmente, los parámetros de los algoritmos de clasificación, por separado.

Cada elemento del vector descrito tiene un rango de opciones propio, de modo que cualquier instancia de este vector representará un modelo válido. Los algoritmos utilizados por la implementación hecha por Escalante et al (2009) se encuentran contenidos en el *toolbox* Clop[1], el cual es descrito en el capítulo de Experimentación y Resultados.

La selección y construcción de una función objetivo es un paso muy importante en la construcción de un algoritmo de optimización. Es por este motivo que la próxima sección se concentra en un análisis de las opciones disponibles para la construcción de una función que permita evaluar un modelo de aprendizaje supervisado, con énfasis en las técnicas evaluadas para este trabajo.

---

[1] Ver http://clopinet.com/CLOP/

2.2. Funciones Objetivo

En el punto anterior se describieron dos algoritmos de optimización, cuya aplicación a la selección de modelo ha sido de importancia para esta investigación. La mayor ventaja asociada a un algoritmo de optimización es la forma en la cual poda el espacio de búsqueda, de modo de encontrar un óptimo evaluando la menor cantidad de soluciones candidatas posible. Como fue mencionado anteriormente, los causantes del largo tiempo de cómputo necesario para encontrar una solución al problema de selección de modelo son dos. El primero es el tamaño del espacio de búsqueda. Este problema se ataca, utilizando un algoritmo de optimización que siga una trayectoria de búsqueda lo más corta posible para encontrar un óptimo. El segundo es el largo tiempo que toma la evaluación de la función objetivo de este problema. Es por este motivo que la construcción de la función objetivo es de gran interés para este trabajo. Existe una amplia investigación alrededor de las técnicas de validación de modelo y las métricas utilizadas para medir la calidad. En los siguientes dos puntos se presenta un breve resumen de estas técnicas y algunas de las ventajas y desventajas presentada por cada una de ellas.

2.2.1. Métrica de Evaluación

En el problema de selección de modelo, el objetivo es buscar aquel que minimice el error de clasificación sobre muestras desconocidas. Para lograr esto, generalmente se entrena el modelo sobre un conjunto de datos y luego se prueba clasificando un conjunto de datos disjunto al anterior. Es importante que este segundo conjunto también tenga sus clases explicitadas, de modo de medir el error incurrido en la clasificación. Ambos algoritmos, listados en la sección anterior, permiten la definición de cualquier métrica de desempeño que logre medir el poder predictivo del modelo entrenado. Entre estas posibles métricas están el error absoluto, error cuadrático, *Balanced Error Rate* (BER), área bajo la curva ROC, etcétera.

En este trabajo se utilizó el BER como métrica de evaluación. La gran ventaja de esta métrica es que toma en consideración la tasa de error de cada clase, lo cual previene la construcción de modelos parcializados (que favorezcan la clase mayoritaria) en conjuntos de datos desbalanceados (donde el número de muestras de cada clase es distinto). Otra ventaja de esta métrica, es que es la utilizada en el *Performance Prediction Challenge* (Guyon et al (2006)), por lo que nos permite comparar nuestros resultados más fácilmente. El cálculo de la métrica BER para un modelo x, puede ser descrito según la siguiente ecuación:

$$BER(x) = \frac{\sum_{i=1}^{n} E_i}{n} \qquad (2.2.1)$$

donde n es el número de clases y E_i es el porcentaje de muestras mal clasificadas de la i-ésima clase.

2.2.2. Método de Validación

Una vez definida la métrica a utilizar para la evaluación de un modelo, es necesario definir la metodología de validación. Esta define los pasos a seguir en cada evaluación de modelo, de modo de conseguir una estimación certera de su poder predictivo. Para este proceso existen numerosas opciones que varían en cuanto a la certidumbre y generalidad de la evaluación, a cambio de un mayor o menor tiempo de cómputo. A continuación, listamos algunas de estas opciones y las mayores ventajas y desventajas de cada una. Para un estudio más detallado de las ventajas y desventajas de cada técnica de validación se recomienda revisar, por ejemplo, Devijver et al (1982), Kohavi et al (1995) o Kim (2009).

Una de las técnicas de validación más simple y común es *Hold Out*. Esta técnica consiste en, simplemente, dividir aleatoriamente el conjunto de datos en dos conjuntos disjuntos. El primero será utilizado para entrenar el modelo. El segundo será utilizado para probarlo, aplicando alguna métrica de evaluación como la explicada en el punto

anterior. Este método es de muy simple implementación y relativamente rápida evaluación, pero tiene un problema fundamental asociado a la definición del porcentaje de los datos que irán al conjunto de entrenamiento y los que irán al conjunto de prueba. Si se dejan demasiados datos para el conjunto de entrenamiento, queda un conjunto de prueba muy reducido. Esto hace que cualquier métrica de evaluación calculada sea propensa a mucho ruido dependiendo de la dificultad de clasificación de los datos que cayeron en el conjunto de pruebas. Si se dejan muy pocos datos en el conjunto de entrenamiento, el modelo sería probado bajo condiciones muy distintas a las que se planea utilizar finalmente. Es importante recordar que el objetivo de este proceso es estimar el poder predictivo que este modelo tendría sobre muestras nuevas, utilizando todos los datos conocidos. La metodología *Hold Out* es usada, generalmente, para obtener una estimación inicial de la calidad de un modelo, pero es demasiado sensible al ruido como para ser la metodología definitiva de evaluación.

Una segunda opción y una de las más utilizadas, es *k-fold Cross Validation*. El procedimiento seguido por esta técnica es el siguiente. En primer lugar se divide el conjunto de datos en k subconjuntos aleatorios disjuntos, llamados *folds*. Luego, se itera, separando uno de los *folds*, entrenando con los otros $k-1$ *folds* y luego probando sobre el *fold* separado. Una vez terminado este proceso, se promedian los k resultados obtenidos. Este proceso tiene la ventaja de que, para un k lo suficientemente grande, el resultado es bastante estable y un buen estimador del desempeño del modelo sobre datos nuevos. La mayor desventaja, es que a medida que aumenta k, el tiempo de evaluación crece.

Una implementación particular de *Cross Validation* es *Leave-One-Out Cross Validation*. Como lo dice su nombre, el proceso consiste en separar únicamente una muestra. Luego, entrenar con el resto y evaluar el desempeño sobre la muestra separada. Este proceso se repite para cada una de las muestras y es equivalente a hacer *Cross Validation* con un k igual al número de muestras (*folds* de tamaño 1). Este proceso es muy largo, especialmente para conjuntos de datos grandes. Pero existen tres grandes

ventajas al considerar un número más grande de *folds*. La primera, como fue mencionada en el párrafo anterior es obtener un resultado más estable al promediar el desempeño obtenido en cada procéso de evaluación. La segunda, es que al aumentar el número de *folds*, el conjunto de entrenamiento es más grande, y por lo tanto, son condiciones más cercanas al escenario que se desea emular. Por último, obtener varias evaluaciones independientes permite hacer una medición estadística del nivel de estabilidad del modelo, lo cual es información muy relevante de la calidad del modelo.

Finalmente, presentaremos un método de validación menos común pero que ha sido de enorme relevancia para este trabajo, el llamado *Repeated Random Sub-Sampling*. El procedimiento seguido por este método es hacer reiteradas validaciones *Hold Out* y promediar los resultados. Este método tiene varias de las ventajas de *Cross Validation*, además de algunas adicionales. En primer lugar, al igual que *Cross Validation*, es bastante estable sobre un número alto de repeticiones, se puede evaluar utilizando un conjunto de entrenamiento grande por lo que se acerca a las condiciones en las que se utilizará finalmente el algoritmo y permite una estimación estadística del nivel de estabilidad de los resultados del modelo. Además, a diferencia de *Cross Validation*, el número de evaluaciones no tiene relación con el tamaño del conjunto de entrenamiento. Este último punto es de enorme importancia en este trabajo ya que, no sólo es posible evaluar el nivel de incertidumbre del resultado obtenido, sino que además, es posible reducir este nivel de incertidumbre evaluando nuevamente el modelo. Esta característica será un factor fundamental en la construcción del algoritmo propuesto PILS.

Cada uno de los métodos presentados utiliza una etapa de muestreo aleatorio de los datos. Este muestreo divide el conjunto de datos en subconjuntos disjuntos. Para cada una de estas técnicas existe una variante estratificada y una no estratificada. La diferencia se expresa en si el muestreo aleatorio fuerza, o no, que cada subconjunto de datos mantenga la misma proporción de elementos de cada clase, que tiene el conjunto de datos original. Aunque requiera de un tiempo de ejecución levemente más largo, en

este trabajo únicamente se consideró la variante estratificada de cada una de estas técnicas ya que ha demostrado ser un medidor mucho más estable.

Generalmente, es considerado válido por la comunidad científica que, al presentar los resultados de un modelo de aprendizaje supervisado, esta se haga usando *Cross Validation* con alrededor de 10 *folds*, ya que éste retorna un valor relativamente estable y repetible. Este tema es de gran relevancia para este trabajo, ya que la técnica seleccionada será ejecutada cada vez que se intente evaluar la función objetivo en cualquiera de los algoritmos de optimización. Estas técnicas de optimización requieren de una evaluación certera de la función objetivo para cada modelo candidato considerado, la cual será evaluada numerosas veces. El enfoque del algoritmo propuesto fue tomar ventaja de las técnicas de validación más rápidas, combinada con la certidumbre de las técnicas más lentas. Esto es posible ya que se hace una evaluación del nivel de incertidumbre de cada modelo evaluado. Una vez hecho este análisis, es posible gastar un mayor de tiempo de procesamiento al disminuir la incertidumbre (a través de la reevaluación), únicamente, para los modelos que el algoritmo estima que tienen una alta probabilidad de ser óptimos.

3. DISEÑO PROPUESTO

De modo experimental, al aplicar paramILS a la selección de modelo, éste demostró utilizar una trayectoria muy efectiva en la búsqueda de un óptimo. El problema que surge es que, aunque el algoritmo logre encontrar un buen modelo evaluando una porción muy pequeña del espacio de búsqueda, esto aún toma demasiado tiempo como para poder ser utilizado en la práctica. El motivo, es que este algoritmo depende de una evaluación certera de la función objetivo para cada solución candidata. Esto se traduce en que requiere de una técnica de validación de baja incertidumbre, como *Cross Validation*. La desventaja de estas técnicas de validación es que toman demasiado tiempo de ejecución en hacer la evaluación de cada modelo.

Por esta razón, este trabajo se enfoca en la construcción de un algoritmo que permita la utilización de un método probabilístico para así trabajar con estimados de la función objetivo en lugar de una evaluación certera de la misma. Este algoritmo fue llamado *Probabilistic Iterative Local Search* (PILS). En este capítulo se describen los principios de este algoritmo de optimización probabilístico. Para lograr esto, se comienza por describir las definiciones y supuestos necesarios para la implementación de esta técnica. Luego, se describe tanto la teoría matemática como la implementación práctica del algoritmo en sí. Además, se explican algunas optimizaciones de implementación incorporadas. Por último, se describe la aplicación de este algoritmo al problema de selección de modelo.

3.1. Definiciones

Como fue mencionado anteriormente, este algoritmo no se encuentra diseñado para trabajar directamente con la función objetivo. En su lugar, se trabaja con una estimación de la misma, que requiera una cantidad considerablemente menor de tiempo de ejecución, a la cual llamamos función estimada. A cambio de esta significativa reducción en tiempo de ejecución, se permite que esta función estimada sea ruidosa. En las ecuaciones presentadas a continuación, se muestra nuestra modelación matemática de

este ruido estocástico. En la ecuación **(3.1.1)**, definimos el valor de la función estimada $\hat{f}(x)$ para una posible solución x, como la suma del valor real de la función objetivo $f(x)$, evaluada en x, más un ruido estocástico que depende de x, al que llamamos e_x.

$$\hat{f}(x) = f(x) + e_x \tag{3.1.1}$$

Para el correcto funcionamiento del algoritmo propuesto, se debe definir la función estimada de modo que un promedio de reiteradas evaluaciones de la misma, tienda al valor de la función objetivo original. En otras palabras, que el ruido estocástico generado en el proceso de estimación, tenga media 0. Esto se puede ver matemáticamente en **(3.1.2)** y **(3.1.3)**.

$$E\,(\,\hat{f}(x)\,) = f(x) \tag{3.1.2}$$

$$E\,(e_x) = 0 \tag{3.1.3}$$

Resulta claro que no cualquier estimación de la función objetivo cumple con esta restricción, pero ésta es de carácter fundamental para el funcionamiento del algoritmo.

3.2. Supuestos

El primer supuesto, y el más relevante, sobre el error estadístico antes llamado e_x dice: Para una solución x determinada (y en este contexto un conjunto de datos determinado también), reiterados llamados a la función estimada $\hat{f}(x)$ contienen observaciones de e_x que son independientes e idénticamente distribuidas entre ellas. En otras palabras, como fue planteado en la ecuación **(3.1.1)**, cada evaluación de la función estimada es la suma del valor de la función objetivo real más variable aleatoria. Este supuesto dice que el componente aleatorio de una evaluación de la función estimada es una variable aleatoria independiente e idénticamente distribuida al componente aleatorio de otras evaluaciones a esta función estimada sobre la misma solución x.

El supuesto descrito en el párrafo anterior es de gran relevancia para la construcción de este algoritmo. El motivo por el cual es tan favorable, es que permite la incorporación del Teorema Central del Limite (TCL) en la construcción del algoritmo propuesto. Para la utilización del TCL es necesario un conjunto de variables aleatorias independientes e idénticamente distribuidas. Como se podrá ver en el punto siguiente, la unión de este supuesto con la definición descrita en **(3.1.3)**, permitirá modelar estadísticamente el nivel de certidumbre que se tiene sobre un conjunto de evaluaciones de la función estimada sobre una solución x.

El segundo supuesto sobre los conjuntos de variables aleatorias mencionados en el párrafo anterior dice: Aunque la varianza asociada al conjunto de variables aleatorias de ruido difiera dependiendo de la solución x evaluada, esta diferencia es pequeña. En otras palabras, si se evalúa reiteradamente la función estimada para dos soluciones x_1 y x_2 diferentes, la varianza asociada al ruido para cada solución va a ser similar.

El supuesto planteado en el párrafo anterior no es absolutamente imprescindible para la implementación del algoritmo propuesto. Pero, después de haber sido descubierto, se incorporó en una mejora al algoritmo, específica para este contexto. Su gran ventaja, es que permite hacer una estimación inicial del nivel de incertidumbre asociada a la medición de calidad de una solución, sin la necesidad de un gran número de observaciones de la misma. Esto es gracias a que se puede usar la varianza promedio que se ha conseguido en la evaluación de otras soluciones como estimador de la varianza de la solución nueva.

El algoritmo aquí descrito considera estos dos supuestos. Los cuales fueron validados experimentalmente, a través de exhaustivas pruebas a lo largo de esta investigación. Es por esto que es posible afirmar que para el contexto que aquí se plantea, son razonables. Es importante recalcar la importancia de validar estos supuestos, ya que son parte fundamental de la teoría matemática sobre la cual se sustenta el algoritmo, por lo que cualquier aplicación de este algoritmo a problemas en un

contexto diferente, requiere de una validación, ya sea teórica o experimental, de los mismos.

3.3. Modelación Estadística del Error

Como ha sido mencionado, el mayor esfuerzo de este trabajo se enfoca en la modelación estadística del ruido asociado a la evaluación reiterada de la función estimada. El modo en el cual el algoritmo hace esto es el siguiente: Para cada solución candidata considerada se tiene un número variable (n_x) de evaluaciones. Gracias a la definición descrita en la ecuación **(3.1.3)**, sabemos que el valor esperado del ruido de este conjunto es 0. Además, gracias al primer supuesto y al TCL, sabemos que es posible modelar el promedio del conjunto de errores como una variable aleatoria que distribuye Normal. Tomando ambas juntas sabemos que el promedio del conjunto de evaluaciones de error distribuye Normal media 0 y una varianza igual a la varianza del conjunto de evaluaciones ${\sigma_x}^2$, dividido por el numero de evaluaciones n_x, como dice **(3.3.1)**.

$$\overline{e_x} \sim N(0, \frac{{\sigma_x}^2}{n_x}) \tag{3.3.1}$$

Es fácil ver, en base a **(3.3.1)** y **(3.1.1),** que podemos modelar el valor de la función objetivo, evaluado en x, como una variable aleatoria Y_x, que distribuye Normal, con una media $f(x)$ y una varianza $\frac{{\sigma_x}^2}{n_x}$, como dice **(3.3.2)**. Gracias al TCL, para un n_x lo suficientemente grande, sabemos que esta distribución se puede estimar como fue planteada en **(3.3.3)**, donde $\overline{f_x}$ es la media y ${s_x}^2$ es el estimador insesgado de la varianza de un conjunto de muestras de $\hat{f}(x)$.

$$Y_x \sim N(f(x), \frac{{\sigma_x}^2}{n_x}) \tag{3.3.2}$$

$$Y_x \sim N(\overline{f_x}, \frac{{s_X}^2}{n_x}) \tag{3.3.3}$$

La ventaja de esta modelación, es que permite comparar las estimaciones calculadas de dos soluciones candidatas. Además, permite cuantificar la probabilidad de estas comparaciones. En la práctica, cada vez que el algoritmo es presentado con dos soluciones candidatas, éste calcula la probabilidad de que la nueva solución sea mejor que la anterior **(3.3.4)** y la probabilidad de que esta nueva solución sea muy similar **(3.3.5)**. Con esta información, es posible decidir si aceptar la nueva solución como mejor, igual, peor o si hay demasiada incertidumbre y, por lo tanto, reevaluar las muestras.

$$P(Y_{x_1} < Y_{x_2}) \tag{3.3.4}$$

$$P(Y_{x_1} - Y_{x_2} < \varepsilon) \tag{3.3.5}$$

3.4. Algoritmo

El algoritmo propuesto se basa en las definiciones hechas por paramILS sobre la meta-heurística ILS. Ese algoritmo demostró, experimentalmente, que sigue una trayectoria muy eficiente a través del espacio de búsqueda para encontrar un óptimo. Su mayor limitación era el tiempo que requería para la evaluación de cada solución candidata. Es esto lo que intenta solucionar el algoritmo propuesto PILS. Para esta implementación se siguen todas las definiciones originales de paramILS, de modo de no modificar la trayectoria de búsqueda del mismo, pero se modifica la función `Mejor`, de modo de poder utilizar estimaciones de la función objetivo. La diferencia de esta función `Mejor` es que compara dos distribuciones de probabilidad en lugar valores certeros (ver **Figura 2**).

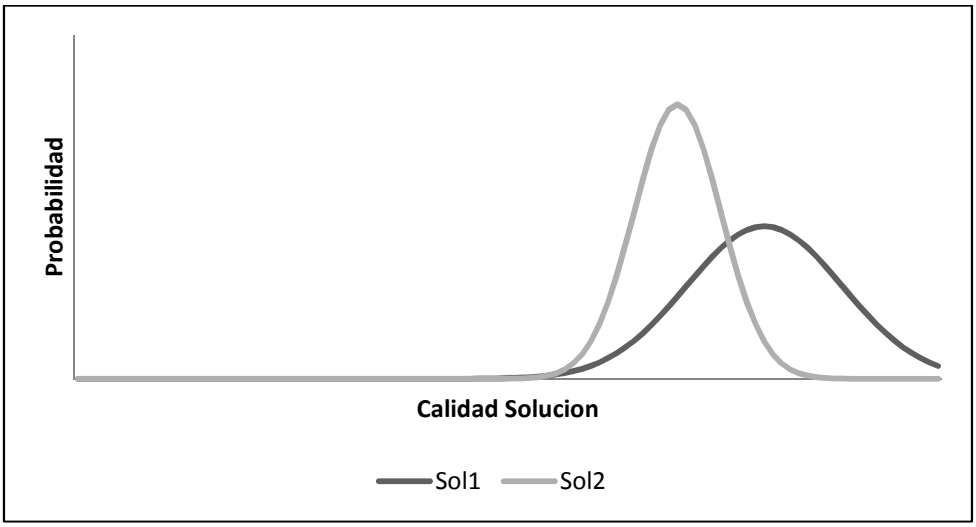

Figura 2: Distribución de probabilidad de dos modelos candidatos.

Algoritmo 5: Función Mejor PILS

```
Parámetros:
      - x1, x2: Soluciones Candidatas Comparadas. Para cada una de
        estas soluciones se tiene además:
      o mu1, mu2: Media de las evaluaciones de x1 y x2.
      o var1, var2: Varianza de las evaluaciones de x1 y x2.
      o n1, n2: Número de veces que se ha evaluado x1 y x2
        respectivamente.
Constantes:
      - SIG: Nivel donde se considera significativa la probabilidad.
        (Por defecto: 0.9 )
      - CUT: Corte donde se asume que es demasiado improbable que la
        solución mejore lo suficiente. (Por defecto: 0.5 )
      - N_MAX: Número máximo de evaluaciones permitidas para una
        solución. (Por defecto: 10)
Retorna:
      Verdadero si x1 es mejor solución de x2, falso en otro caso.

Mejor (x1, x2)
{
      loop
      {
            [pm, pi] = Probabilidades_Mejor (x1, x2);
            if pm >= SIG
                  return true;
            if pi >= SIG
                  return var1<var2;
            if pm < CUT
                  return false;
            if min(n1, n2) >= N_MAX
                  return mu1>mu2;
            if n1<n2
                  evaluar (x1);
            else
                  evaluar (x2);
      }
}
```

Esta implementación de la función `Mejor` utiliza la función auxiliar `Probabilidades_Mejor`. El objetivo de esta función auxiliar es hacer un cálculo estadístico en base a la información disponible, retornando dos probabilidades. La primera, llamada pm, es la probabilidad de que x1 sea mejor que x2 y es equivalente a la expresión **(3.3.4)**. La segunda, llamada pi, es la probabilidad de que x1 sea igual (o muy similar) a x2 y es equivalente a la expresión **(3.3.5)**. Una vez calculada esta información, el algoritmo toma una de las siguientes opciones:

- Si la probabilidad de que la solución nueva x2 sea mejor que la solución antigua x1 es significativa (mayor que SIG), se retorna verdadero.
- Si la probabilidad de que estas soluciones sean iguales (o muy similares) es significativa, se retorna verdadero en caso que la varianza de la nueva sea menor que la de la solución antigua, falso en otro caso. Ya que si son iguales, es preferible una solución más estable.
- Si la probabilidad de que la solución nueva llegue a ser mejor que la solución antigua es demasiado baja (menor que CUT), se asume que ésta nunca mejorará lo suficiente y se retorna falso.
- Si ambas soluciones ya alcanzaron el número máximo de evaluaciones y aún no hay una clara ventaja de una, se elige la que tenga la mayor media.
- Si no fue posible tomar ninguna de las decisiones anteriores, se reevalúa la solución candidata que tenga menor número de evaluaciones (disminuyendo así su incertidumbre) y se vuelve a iterar.

3.5. Criterio de Término

Los algoritmos basados en ILS, presentados en este texto, tienen en común que en todo momento mantienen un registro de la mejor solución encontrada hasta ese instante. Al revisar la definición de ILS, es posible ver la necesidad de fijar un criterio de término para el ciclo central del algoritmo. Este criterio de término define cuándo el algoritmo se encuentra satisfecho con la solución que ha encontrado y detiene su

ejecución. La implementación hecha por Hutter et al (2009) en paramILS, no incluye una definición de un criterio de término para su algoritmo. Es a causa de esto, que fue necesaria la definición de un criterio de término propio para PILS.

Resulta importante recalcar que la implementación de la función `Mejor` definida para PILS, se encuentra sesgada a la solución actual. En otras palabras, para definir un vecino de la solución actual como mejor (y como consecuencia, saltar a él repitiendo el proceso), éste debe demostrar ser significativamente mejor que la solución encontrada. Esto se debe a dos motivos. El primero, es que cada salto conlleva un alto costo de tiempo asociado. El segundo, es que debido al ruido asociado a la función estimada, es fácil que dos soluciones de igual calidad tengan promedios levemente distintos. Gracias a este sesgo asociado a la solución actual, el resultado al final de cada iteración es bastante estable. En otras palabras, el óptimo cambia, únicamente, cuando se encuentra una solución significativamente diferente y no a causa del ruido.

Debido a la característica mencionada en el párrafo anterior, fue posible implementar varios criterios de término con buenos resultados. Los mejores resultados se lograron evaluando la estabilidad de la solución encontrada. Esta técnica simplemente evalúa cuántas iteraciones han pasado desde que la solución óptima cambió. Si este número de iteraciones supera un umbral predefinido, se detiene la ejecución del algoritmo. Fue posible ver empíricamente que, aun utilizando un umbral bajo (por ejemplo 2), esta técnica retorna resultados muy buenos.

Aunque el criterio de término mencionado demostró retornar buenos resultados, la implementación utilizada en las pruebas maneja un número fijo de iteraciones como criterio de término. El motivo de esta implementación, es que resulta en un tiempo de ejecución más estable. Además, para permitir una comparación más justa entre este algoritmo y otras técnicas, se requería de un parámetro que otorgue mayor control sobre, tanto el tiempo de procesamiento del algoritmo, como la calidad del modelo retornado

por el mismo. Es por esto, que todos los resultados de este trabajo, son obtenidos en base a un número predefinido de iteraciones.

3.6. Optimizaciones de Implementación

Anteriormente, se explicó la estructura general del algoritmo propuesto PILS. En este punto describimos algunas optimizaciones de implementación, que aunque no son completamente necesarias para la implementación de este algoritmo, demostraron un ahorro considerable de tiempo de ejecución.

3.6.1. Estimación de la varianza

El algoritmo descrito requiere de un estimador insesgado de la varianza, al que llamamos $s_x{}^2$. El problema es que ahora se requiere de al menos dos observaciones para una estimación de la varianza. Un análisis empírico demostró que el problema era aún peor, ya que el cálculo de la varianza utilizando únicamente dos observaciones era un estimador muy pobre de la misma. Pero, gracias al supuesto mencionado anteriormente, que las varianzas eras distintas pero similares entre soluciones, es posible utilizar la información calculada de muestras de otras soluciones para estimar la varianza de una nueva (sobre la cual no hay suficientes muestras). Si suponemos que existe una variable global, llamada $VarGlobal$, que contiene el promedio de las varianzas calculadas hasta el momento, y que llamamos $VarMuestral_x$ a la varianza que se ha estimado a partir de las n_x muestras de la solución actual x, podemos definir la varianza estimada como un promedio ponderado entre la varianza global y la estimada a partir de la muestra **(3.5.1)**.

$$s_x{}^2 = \frac{VarGlobal*(N\_MAX-n_x)+VarMuestral_x*(n_x-1)}{N\_MAX-1} \quad (3.5.1)$$

A través de experimentación se vio que la varianza global depende, fuertemente, del conjunto de datos utilizado y del modo en el que se defina la función estimada. Dado que es imposible estimar esta variable a priori, se hace necesaria una técnica de estimación empírica de la misma. Se consideró, con buenos resultados, hacer un

muestreo aleatorio de soluciones antes de la ejecución del algoritmo de modo de obtener un estimado inicial. El problema con este enfoque es que sacrifica demasiado tiempo de cómputo. En su lugar, se desarrolló una metodología que actualiza este valor a medida que el algoritmo busca el óptimo. Para esta solución, es necesario que el usuario defina una varianza estimada a priori, $VarPrior$ y un peso para esta varianza $PesoPrior$, equivalente al nivel de certidumbre que se tiene sobre el prior. Luego, cada vez que se reevalúan soluciones candidato, la información de varianza asociada a la misma actualiza el estimador global, como se puede ver en la ecuación **(3.5.2)**. La experimentación ha demostrado que esta metodología de varianza global converge rápidamente y que es un muy buen estimador de la varianza promedio.

$$VarGlobal = \frac{VarPrior * PesoPrior + \sum_i VarMuestral_i * (n_i - 1)}{PesoPrior + \sum_i (n_i - 1)} \quad (3.5.2)$$

3.6.2. Almacenamiento de los datos

La trayectoria de búsqueda seguida tanto por paramILS como por PILS, salta en cada iteración a puntos nuevos aleatorios y comienza nuevamente una búsqueda de un óptimo local cercano. En este proceso, es posible tener la necesidad de reevaluar alguna solución que ya fue considerada anteriormente. Es por esto que resulta ventajoso almacenar la información ya extraída de cada solución, de modo de no requerir empezar el proceso nuevamente. Para almacenar esta información, el algoritmo mantiene una tabla de hash, indexada de acuerdo a la solución, que almacena toda la información útil que se ha extraído hasta ahora de cada solución evaluada.

Almacenar el resultado de cada una de las evaluaciones de una solución requiere de un gasto innecesario de memoria. Además, recalcular la media y varianza del subconjunto cada vez que se desea comparar, consume demasiado tiempo. Es por este motivo que se construyó una estructura que permita almacenar esta información de modo eficiente. En la ecuación **(3.5.3)** se presenta la estructura de datos que almacena la tabla de hash para cada solución evaluada,

$$Información X = < S, S_2, n > \quad (3.5.3)$$

donde S es la suma directa de las evaluaciones de la función estimada para esa solución, S_2 es la suma de estos valores al cuadrado y n es el número de evaluaciones que se han hecho para esta solución candidata. Las ventajas de almacenar únicamente esta información son tres. En primer lugar, requiere de un menor espacio de memoria, independiente del número de evaluaciones de la función estimada que se hagan. En segundo lugar, es de muy rápida actualización al hacer una evaluación nueva. Y por último, con esta información es posible calcular muy rápidamente la media y varianza del conjunto de evaluaciones hechas.

3.7. Aplicación a la selección de modelo

El algoritmo descrito en este capítulo es un algoritmo de optimización aplicable a problemas de diversos contextos, siempre que se puedan cumplir las definiciones requeridas para el correcto funcionamiento del mismo. Habiendo dicho eso, es importante recalcar el objetivo inicial para el que se diseñó PILS. Este objetivo es encontrar, de modo eficiente, un modelo que logre ajustarse a un conjunto de datos, de modo de clasificar nuevas muestras del conjunto con el menor error posible. Para aplicar este algoritmo al contexto de la selección de modelo, es necesaria una definición del modo de representación de cada modelo factible y la construcción de una función estimada que cumpla con los requisitos del algoritmo. En este punto se dará un breve resumen de estas dos componentes esenciales para el uso de este algoritmo en este problema.

En primer lugar, es necesario definir la representación del espacio de búsqueda que utilizará el algoritmo. Esta representación, se basó en la utilizada por Escalante et al (2009) en el algoritmo PSMS. Tanto PSMS como PILS, utilizan un vector numérico para representar cada modelo. Este vector puede ser descrito del siguiente modo:

- Los primeros elementos del vector representan los métodos de preprocesamiento. Estas son variables binarias, ya que se permite que un modelo contenga varios métodos de preprocesamiento. Por ejemplo, si se consideran tres métodos de preprocesamiento diferentes, éste será representado por tres elementos del vector.
- Los siguientes son los parámetros de los algoritmos de preprocesamiento mencionados anteriormente.
- Luego, una variable entera que representa el algoritmo de selección de variables utilizado. Esto se debe a que sólo es necesario un algoritmo de selección de variables por modelo.
- Los siguientes, son los parámetros de los algoritmos de selección de variables mencionados. En caso que se repitan parámetros entre algoritmos distintos, éstos son agrupados en un parámetro común. Por ejemplo, existe un solo parámetro para definir el número máximo de variables, que es utilizado por todos los algoritmos de selección,
- Una variable entera define qué algoritmo de clasificación se utilizará.
- Finalmente, los parámetros de los algoritmos de clasificación se agregan. En este caso también se agrupan parámetros iguales en un parámetro común.

Esta representación es muy similar a la utilizada por PSMS. La mayor modificación hecha a la misma es la agrupación de parámetros (utilizados por algoritmos distintos) en parámetros comunes. Existen dos motivaciones importantes para esta agrupación. La primera es que retorna mejores resultados. Por ejemplo, al utilizar un método de selección de variables, se debe fijar el número de variables a seleccionar. Al utilizar las distintas técnicas de selección, probablemente, retornarán conjuntos similares de variables a elegir. Es por esto que los números óptimos de variables de cada técnica suelen ser similares. La segunda motivación, es que ahorra tiempo de cómputo. El algoritmo de Escalante et al (2009) tiene la habilidad de saltar de una solución a una muy alejada, si la velocidad en esa dimensión es grande. Mientras que para PILS, el movimiento debe ser hecho siempre desde una solución a su vecino. Además, PILS

depende de lograr definir una solución como un óptimo local y después de compararlo con todos sus vecinos. Es a causa de esto que aumentar mucho el número de vecinos tiene como consecuencia un aumento considerable en el tiempo de cómputo.

El segundo punto que requiere ser definido es la función estimada. Esta es, quizás, la característica más importante en cualquier implementación de este algoritmo. Como fue discutido en el capítulo de Marco Teórico, la técnica utilizada para la evaluación de un modelo de clasificación se puede dividir en dos componentes. El primer componente es la métrica de evaluación usada. En esta implementación de PILS se utilizó BER. Esta métrica tiene la ventaja de medir el desempeño de cada clase individualmente, lo cual retorna un mejor resultado cuando el conjunto de datos no está balanceado (tiene distinto número de muestras de cada clase). El segundo componente, es la técnica de validación usada. Si se desea construir una función objetivo que retorne una medición confiable de la calidad del modelo evaluado, comúnmente, se utilizaría *Cross Validation*. Dado que esto tomaría demasiado tiempo, y que el algoritmo propuesto se encuentra diseñado para manejar la incertidumbre, se utilizó *Hold Out* como función estimada. Como fue descrito en el capítulo de Marco Teórico, el promedio de un número alto de validaciones *Hold Out* se llama *Repeated Random Sub-Sampling*. La ventaja de *Repeated Random Sub-Sampling* es que, bajo un número alto de repeticiones, converge al mismo valor que *Cross Validation*, con un k lo suficientemente grande. Es gracias a esto que definimos nuestra función estimada usando *Hold Out*, como el estimador ruidoso de una función objetivo usando *Cross Validation*.

4. EXPERIMENTOS Y RESULTADOS

A lo largo de este trabajo se realizaron numerosos experimentos con el fin de revisar si los supuestos hechos eran válidos y comprobar la calidad del algoritmo propuesto. Un primer subconjunto de estos resultados fue publicado en Cortazar y Mery (2011). En esta sección se exponen los experimentos enfocados a la medición del desempeño del algoritmo. Estos se pueden separar en tres etapas. La primera fue una prueba de concepto. Esta intentaba verificar la calidad del algoritmo bajo condiciones controladas. La segunda fue una comparación entre el algoritmo propuesto versus paramILS. Esta tenía como objetivo validar las modificaciones hechas sobre el algoritmo base. Por último, se comparó el algoritmo propuesto contra PMSM. Esta prueba demuestra el aporte hecho por el algoritmo propuesto, al compararse contra un algoritmo de selección de modelo validado por la comunidad científica.

4.1. Prueba de Concepto

Como fue mencionado anteriormente, esta prueba tiene como objetivo un análisis del comportamiento del algoritmo bajo condiciones controladas. Para lograr esto, es necesaria la ejecución reiterada de los algoritmos, comparándolos bajo métricas válidas. Fue necesaria la creación de una función objetivo artificial que permita ser evaluada de manera certera y rápida.

4.1.1. Diseño del experimento

Como fue mencionado, para esta simulación, fue necesaria la construcción de una función objetivo artificial. El proceso de construcción de la función objetivo artificial fue el siguiente:

1. En primer lugar, se definió el tamaño del espacio de búsqueda. Se probó con varias configuraciones, con resultados equivalentes. En este texto se muestran los datos con un espacio de búsqueda de 5 parámetros, cada uno de los cuales podía

tomar 10 valores, lo cual resulta en un espacio de búsqueda de 5^{10} soluciones candidatas.

2. Luego, se construyó la función objetivo como una mezcla de gaussianas, de modo de construir una función objetivo con varios óptimos locales. Los parámetros de cada gaussiana eran generados de modo aleatorio. Se hicieron pruebas con varias configuraciones, con resultados similares. En este texto se presentan los resultados usando la suma de 5 gaussianas.
3. Por último, dado que el contexto de este problema requiere de algoritmos de programación lineal, la función objetivo era discretizada sobre el espacio de búsqueda definido.

La función objetivo artificial descrita anteriormente representaría los valores obtenidos a través de una evaluación certera de un modelo candidato, por lo que fue denominada Función Determinística. Para el uso del algoritmo propuesto, es necesaria la implementación de una función estimada, la cual sacrifica certidumbre a cambio de una reducción considerable en el tiempo de ejecución. Esta función ruidosa fue denominada Función Aleatoria y se construyó siguiendo los siguientes pasos.

1. En primer lugar, se construye una matriz con dimensiones iguales a las del espacio de búsqueda definido, la cual fue llamada matriz de ruido, ya que representa las varianzas del ruido asociado a cada solución candidata.
2. Luego, se llena la matriz de ruido con valores aleatorios. En este texto se presentan los resultados obtenidos usando un muestreo de una distribución Normal con media 0.3 y desviación estándar 0.001. Con esto se consigue que cada solución candidata tenga un ruido con varianza similar pero no idéntica.
3. Por último, en cada evaluación de la Función Aleatoria, se toma como base el valor de la Función Determinística para esa solución, y se le suma un valor aleatorio con distribución Normal de media 0 y varianza correspondiente a su posición en la matriz de ruido.

Con la definición de estas dos funciones, es posible hacer una evaluación exhaustiva del desempeño de PILS, comparándolo con el de paramILS bajo un escenario controlado. Los experimentos realizados fueron sobre tres configuraciones diferentes de algoritmos. Primero, paramILS usando la Función Deterministica. Este experimento debería ser equivalente a los resultados obtenidos utilizando la implementación directa de paramILS. Segundo, paramILS usando la Función Aleatoria. Este experimento tiene por objetivo, demostrar que paramILS no logra resultados aceptables con funciones objetivo ruidosas. Por último, PILS usando la Función Aleatoria. Esta es una configuración equivalente al uso común que debería tener PILS.

Cada una de las configuraciones fue probada reiteradamente, variando el número de iteraciones del algoritmo desde 1 hasta 36 de modo de ver como progresa su desempeño. Además, dado que estos algoritmos tienen una componente azarosa importante, los resultados expuestos en este texto son el promedio de 1000 ejecuciones de cada opción.

En esta etapa se definieron dos métricas para la evaluación de cada configuración de algoritmo. En primer lugar, se definió una métrica de calidad booleana, donde una ejecución puede ser considerada un éxito o un fracaso. El éxito es obtenido, únicamente, cuando el algoritmo retorna una solución que se encuentra entre las mejores 0.1% del conjunto de soluciones candidatas. En segundo lugar, se registra en número de llamados a la función objetivo, de modo de hacer una estimación del tiempo necesario para cada algoritmo. Resulta importante recalcar que el número de evaluaciones no es comparable entre funciones distintas, ya que, por definición, una evaluación de la Función Aleatoria representa un tiempo de ejecución considerablemente menor que una de la Función Determinística.

4.1.2. Resultados Experimento

En primer lugar, revisando la **Tabla 1**, es posible hacer una comparación entre el número de ejecuciones necesarias para paramILS versus las necesarias para PILS, para

obtener desempeños similares. Estos datos nos permiten deducir dos observaciones de importancia para este trabajo. La primera observación, es que PILS efectivamente es capaz de resolver el problema de optimización, aun utilizando una función objetivo ruidosa. Estos resultados se repitieron a lo largo de numerosas pruebas bajo distintas configuraciones de la función objetivo artificial. Mientras que el ruido asociado a la función estimada no fuese considerablemente más alto que la varianza misma de la función objetivo, PILS fue capaz de encontrar desempeños comparables con el de paramILS. La segunda observación, es que PILS requiere de un número más alto de evaluaciones de su función objetivo que paramILS. Esto era de esperarse viendo el diseño del algoritmo en sí. Pero, debido a que es posible construir una función que requiere de un tiempo de evaluación considerablemente menor, el algoritmo aún resulta en un tiempo de ejecución menor. Por ejemplo, la **Tabla 1** fue construida usando funciones que emulaban el nivel de ruido visto experimentalmente entre una función *Cross Validation* con 5 *folds* versus una usando *Hold Out* con 20% de los datos para el entrenamiento. En este ejemplo, PILS requiere de alrededor del doble de evaluaciones de una función que tomaría aproximadamente un quinto del tiempo, por lo que debería ser capaz de resolver el mismo problema en alrededor de un 40% del tiempo que requeriría paramILS.

Tabla 1: Comparación del número de evaluaciones de la función objetivo de paramILS y de PILS para desempeños similares. Es importante notar que PILS está usando una función ruidosa, mientras paramILS está usando una función con certidumbre.

paramILS		**PILS**	
% Éxito	Evaluaciones F. Deterministica	% Éxito	Evaluaciones F. Aleatoria
76,9	527	78,5	1076
82,0	628	83,5	1203
85,1	727	86,4	1332
90,8	838	90,0	1685
99,0	1419	97,0	2867

La tabla anterior muestra que PILS es, efectivamente, capaz de resolver el problema de optimización utilizando una función ruidosa. Pero, también muestra que requiere de un número mucho mayor de evaluaciones de la misma. Es por esto que es necesario revisar el desempeño que conseguiría paramILS, utilizando una función ruidosa. En la **Figura 3**, es posible ver los desempeños de paramILS y de PILS bajo la misma función ruidosa utilizada por en la **Tabla 1**. Se puede apreciar que, al aumentar el número de iteraciones de paramILS, el desempeño del algoritmo no mejora como si lo hace PILS. Esto se debe a que paramILS no está construido para considerar el ruido, por lo que logra aciertos en forma casi aleatoria.

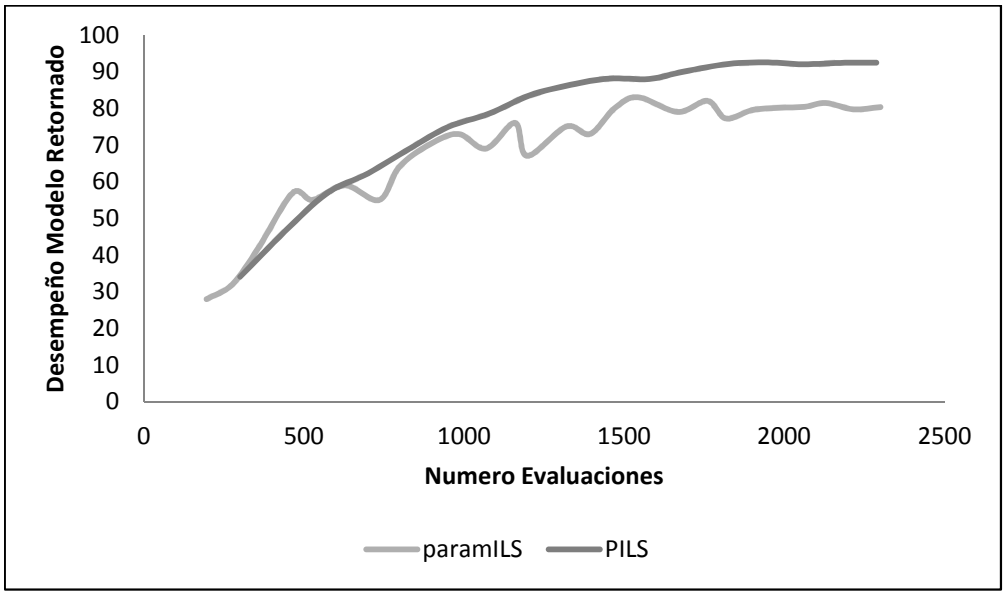

Figura 3: Desempeño vs Número de evaluaciones hechas por paramILS y PILS. Ambas utilizando la función ruidosa bajo distintos números de iteraciones.

Gracias a este experimento, fue posible verificar las ventajas que presenta PILS para el problema de optimización propuesto, además de otorgar un ambiente controlado en donde se pudo calibrar el algoritmo y probar sus limitaciones. También, se demostró la necesidad de un algoritmo con este enfoque para problemas de este tipo, al confirmar las limitaciones de algoritmos como paramILS ante este escenario.

4.2. PILS versus paramILS

En la prueba de concepto se compararon los desempeños de PILS con los de paramILS bajo un escenario simulado. Luego, es necesario hacer una comparación de los algoritmos en el contexto específico del trabajo, el problema de selección de modelo. El objetivo de este experimento es revisar las ventajas del algoritmo propuesto por sobre su algoritmo base, de modo de validar las modificaciones hechas.

4.2.1. Toolbox Balu

Para la implementación de un algoritmo de selección de modelo es necesario definir, en primer lugar, el conjunto de algoritmos que serán el espacio de búsqueda del mismo. Para esto se utilizaron las herramientas disponibles a través de Balu[2]. Balu es un conjunto de herramientas, en Matlab, enfocadas a visión por computador, reconocimiento de patrones y procesamiento de imágenes. Para este trabajo, se utilizaron sus algoritmos de preprocesamiento, selección de variables y clasificación.

La definición del espacio de búsqueda se realizó de la misma manera tanto para paramILS como para PILS, y se puede separar en tres grupos. En primer lugar, algoritmos de preprocesamiento. Estos algoritmos limpian, transforman o reordenan los datos con el objetivo de facilitar la clasificación. En el grupo de preprocesamiento se consideró cualquier combinación de los siguientes algoritmos:

- PCA: Proceso en el que se extraen las componentes principales de los datos y se agregan como variables nuevas. Estas variables contienen la misma información provista por los datos originales, pero su transformación puede retornar una visión más clara de la misma.
- Normalización: Normalización de los datos. Esta normalización se puede efectuar haciendo una transformación lineal de cada variable al rango $[0,1]$ o transformando de modo que cada variable se ajuste a una distribución normal de media 0 y varianza 1.
- Limpieza: Proceso de eliminación de variables altamente correlacionadas.

El segundo grupo es el de selección de variables. En este caso se consideraron dos algoritmos. Cada uno de estos algoritmos tiene, además, un parámetro de número de variables a seleccionar. Los algoritmos de selección de variables considerados son:

---

[2] Ver http://dmery.ing.puc.cl/index.php/balu/

- Rank: Ordena las variables de acuerdo a un criterio de separabilidad, en este caso, un T-test.
- SFS: Selecciona secuencialmente las variables calculando, en cada paso, la ganancia. En este caso se utilizó el discriminante de Fisher como métrica de esta ganancia.

El último grupo es el de clasificación. En este grupo se consideraron seis clasificadores. Para algunos de éstos se consideraron sus parámetros, en otros casos se consideraron los valores por defecto de los mismos. Los algoritmos de clasificación considerados son:

- Dmin: Hace la clasificación de acuerdo la mínima distancia euclidiana.
- Mahalanobis: Hace la clasificación de acuerdo a la mínima distancia Mahalanobis.
- LDA: Clasificador en base a análisis de un discriminante lineal.
- QDA: Clasificador en base a análisis de un discriminante cuadrático.
- KNN: Clasificación en base a k vecinos cercanos. Para este experimento se consideró k igual a 5.
- SVM: Clasificación en base a maquinas de soporte de vectores. Se consideraron 5 opciones de kernel (lineal, cuadrático, polinomial, *Gaussian Radial Basis, Multilayer Perseptron*)

Balu incluye una herramienta de selección de modelo llamada Bcl_gui. Esta herramienta permite seleccionar un conjunto de posibles algoritmos de selección de variables y de clasificación con los que hace una búsqueda exhaustiva de los modelos candidatos. Se consideró hacer una comparación entre paramILS, PILS y Bcl_gui pero, dado el reducido número de parámetros considerados como espacio de búsqueda de Bcl_gui, todos los algoritmos descritos evaluarían el espacio de búsqueda completo. En su lugar, se implemento una búsqueda exhaustiva que maneja el mismo espacio de búsqueda de este experimento. Pruebas preliminares retornaron un resultado trivial: El

tiempo necesario para una búsqueda exhaustiva era considerablemente mayor (entre 5 y 60 veces) que el necesitado por los algoritmos de optimización aquí probados. A causa de esto, no se continúo en esa línea de pruebas.

4.2.2. Diseño del experimento

El diseño de este experimento fue bastante simple. La representación del espacio de búsqueda de ambos algoritmos es idéntica. La única diferencia es la función objetivo considerada por cada algoritmo. El objetivo en esta etapa de experimentación era maximización del desempeño del modelo retornado. Es por esto que ambas funciones objetivo utilizan el desempeño como métrica de calidad. La distinción es que paramILS utiliza una técnica de validación de *Cross Validation* con 5 *folds*, mientras que PILS utiliza *Hold Out* con un 90% de los datos como conjunto de entrenamiento. Se probaron funciones estimadas que hacían muestreo aleatorio del conjunto de datos, para así entrenar y probar con menos datos, pero provocaba una perdida en calidad demasiado grande en comparación a la ganancia en tiempo de ejecución.

El segundo punto a definir para este experimento son los conjuntos de datos sobre los cuales se harán las pruebas. Para esto se utilizó, en primer lugar, un conjunto de características extraídas de radiografías de salmones llamado *fishbones* (Mery et al (2011)). Luego, se agregaron algunos de los conjuntos de datos provistos por el *Performance Predicion Challenge* (Guyon et al (2006)). Un resumen de las características de estos conjuntos se puede ver en la **Tabla 4**. Para poder realizar numerosas pruebas sobre cada conjunto de datos, se hizo un muestreo estratificado de 1000 muestras de cada conjunto de datos y se trabajó únicamente con ellas.

Fueron dos las métricas de calidad utilizadas para cada algoritmo. En primer lugar, se consideró el tiempo que requería cada opción en su búsqueda de modelo. En segundo lugar, se hizo una medición del desempeño del modelo entregado por cada algoritmo. Para la medición del desempeño del modelo, se hace una validación *Cross Validation* con 10 *folds*, sobre el mismo conjunto de datos, con el modelo retornado.

Además, de modo de considerar el factor aleatorio de cada algoritmo, se ejecutó reiteradamente cada algoritmo sobre cada conjunto de datos. En este texto se presenta un promedio de los resultados obtenidos sobre cada conjunto de datos.

4.2.3. Resultados Experimento

En la **Tabla 2**, se muestra el desempeño del modelo obtenido y el tiempo requerido para encontrarlo de paramILS y PILS, sobre el muestreo de cada conjunto de datos. Revisando estos datos se puede apreciar que el desempeño conseguido por PILS es menor que el de paramILS, pero esta diferencia no alcanza a ser significativa para ninguno de los conjuntos de datos considerados. Por otro lado, al ver el tiempo requerido para la búsqueda de ese modelo, podemos ver una enorme ventaja de PILS por sobre paramILS. En estos ejemplos, PILS demora entre un 5% y un 20% del tiempo requerido por paramILS para conseguir los mismos resultados.

La gran diferencia en tiempo de ejecución requerido por PILS y paramILS, puede ser explicada principalmente por dos factores. El primero es el cambio de función objetivo. Como se ha mencionado reiteradamente a lo largo de este texto, el enfoque de PILS es lograr un ahorro de tiempo, en base a utilizar una función estimada de muy rápida ejecución. El segundo factor es la trayectoria de búsqueda. El motivo por el que PILS basó su construcción en paramILS, era para rescatar la trayectoria de búsqueda que tenía el mismo, ya que ésta era muy eficiente. Pero, en casos como éste, es posible que la trayectoria seguida por PILS sea más eficiente que la de paramILS. El causante de esto es que aunque *Cross Validation* retorne un valor con un alto nivel de certidumbre, éste aún puede ser variable en un porcentaje no significativo. Esta variación, puede provocar que paramILS haga más saltos de solución en solución que los necesarios. Debido a que para la ejecución de este algoritmo, un salto puede ser muy costoso, el tiempo de ejecución puede verse severamente castigado.

Tabla 2: Comparación de desempeño del modelo encontrado y tiempo necesario para encontrarlo, entre paramILS y PILS, sobre un muestro de 1000 datos distintos conjuntos.

Conjunto de Datos	**paramILS**		**PILS**	
	Desempeño	Tiempo (Minutos)	Desempeño	Tiempo (Minutos)
Fishbones (1k)	0,910	2259	0,890	500
ADA (1k)	0,831	988	0,821	74
SYLVA (1k)	0,990	1953	0,983	342

En resumen, es posible ver que existe un sacrificio en el desempeño al utilizar una técnica probabilística como la propuesta en este trabajo, pero este sacrificio es muy pequeño en comparación con la reducción en tiempo de ejecución obtenido. Además, es posible ejemplificar el costo de tiempo, antes descrito, necesario para la resolución de un problema de selección de modelo, motivo que justifica la enorme necesidad de una técnica que sea capaz de encontrar un modelo rápidamente. Finalmente, en base a estos resultados, es posible validar las modificaciones hechas al algoritmo de búsqueda paramILS.

4.3. PILS versus PSMS

Finalmente, fue necesaria una comparación del algoritmo propuesto versus un algoritmo estado del arte en selección de modelo. Para esto, se seleccionó PSMS como punto de comparación. Como fue mencionado, PSMS tuvo un excelente desempeño en la competencia del *Performance Prediction Challenge*. Su desempeño fue tan bueno, que se ha incorporado dentro del *Challenge Learning Object Package* (CLOP), provisto por la competencia. En esta comparación, se utilizará la implementación provista de PSMS.

4.3.1. Challenge Learning Object Package (CLOP)

Para la implementación de cualquier algoritmo de selección de modelo, es necesaria una batería de algoritmos de preprocesamiento, selección de variables y clasificación. Para este experimento, se hizo uso de un paquete de herramientas de aprendizaje de máquina de dominio público, llamado *Challenge Learning Object Package* (CLOP). CLOP es provisto por la competencia del *Performance Prediction Challenge* e incluye, además de los algoritmos antes mencionados, una la implementación de PSMS hecha por Escalante et al (2009).

En la **Tabla 3** se presenta un breve resumen de los distintos algoritmos de preprocesamiento, selección de variables y clasificación. Una de las ventajas de CLOP es que permite encadenar una secuencia de algoritmos en un nuevo objeto. Para este experimento se considera como espacio de búsqueda, cualquier cadena que incluya un conjunto (puede ser vacío) de algoritmos de preprocesamiento, un algoritmo de selección de variables y un algoritmo de clasificación, cada uno de los algoritmos con sus respectivos parámetros definidos.

Tabla 3: Resumen de los algoritmos de Preprocesamiento (Pre), Selección de variables (Sel) y Clasificación (Clas) provistos por CLOP, junto a sus parámetros.

Objeto	Tipo	Parámetros	Descripción
normalize	Pre	center	Normalizacion
standarize	Pre	center	Estandarizacion
shift-scale	Pre	TAKElog	Shift and Scale
Ftest	Sel	Fmax, Wmin, Pval, FDRmax	Ranking (f-statistic)
Ttest	Sel	Fmax, Wmin, Pval, FDRmax	Ranking (t-statistic)
Aucfs	Sel	Fmax, Wmin, Pval, FDRmax	Ranking (criterio AUC)
odds-ratio	Sel	Fmax, Wmin, Pval, FDRmax	Ranking (Odds Ratio statistic)
Relief	Sel	Fmax, Wmin, Kman	Ranking Relief
Rffs	Sel	Fmax, Wmin	Selección usando RF
Svcrfe	Sel	Fmax	Eliminación recursiva (SVM)
Pearson	Sel	Fmax, Wmin, Pval, FDRmax	Ranking (Pearson coef)
Zfilter	Sel	Fmax, Wmin	Ranking (Heuristic Filter)
Gs	Sel	Fmax	SFS (Gram-Shmidt)
s2n	Sel	Fmax, Wmin	Ranking (Signal to Noise)
pc-extract	Sel	Fmax	Extracción usando PCA
Zarbi	Clas	-	Clasificador Lineal
Naive	Clas	-	Naive Bayes
klogistic	Clas	-	Kernel Logistic Regretion
Gkridge	Clas	-	Kridge Generalizado
logitboost	Clas	units number, shrinkage, depth	Boosting usando arboles
Neural	Clas	units number, shrinkage, maxiter, balance	Red Neuronal
Svcrfe	Clas	shinkage, coef0, degree, gamma	Clasificador SVM
Kridge	Clas	shinkage, coef0, degree, gamma	Kernel Ridge Regretion
Rf	Clas	units number, balance, mtry	Random Forest
Lssvm	Clas	shinkage, coef0, degree, gamma, balance	Kernel Ridge Regretion

4.3.2. Diseño del experimento

El *Performance Prediction Challenge* provee, además del paquete de herramientas CLOP, varios conjuntos de datos de distintas características. En la **Tabla 4** se presenta un resumen del tamaño de cada conjunto. Cada uno de estos conjuntos de datos se encuentra dividido en tres partes. La primera, es llamada conjunto de entrenamiento, que incluye la mayoría de los datos con su clase explicitada. La segunda es llamada conjunto de validación, que incluye menos datos con su clase explicitada. Por último, la tercera es llamada conjunto de prueba, que incluye menos datos sin su clase. El motivo de esta división es que los concursantes entrenaban sus modelos sin conocer la calidad sobre el conjunto de pruebas. Debido que la clase del conjunto de pruebas nunca ha sido divulgada, no será tomado en cuenta para este experimento.

Tabla 4: Características de los conjuntos de datos provistos por el Performance Prediction Challenge, usados por este trabajo. (El número de muestras se refiere al conjunto de entrenamiento).

Conjunto de Datos	**Número de Variables**	**Número de Muestras**
ADA	48	4147
GINA	970	3153
HIVA	1617	3845
SYLVA	216	13086

Este experimento se diseñó con el objetivo de efectuar una comparación justa entre PILS y PSMS. Para eso, se usó la implementación original de PSMS, utilizando la misma configuración que presentaron para la competencia. Esta configuración define una población de 10, un número de iteraciones igual a 100 y una función objetivo que

mide el BER usando *Cross Validation* con 5 *folds*. La única modificación es que el algoritmo original permite la opción de crear un ensamble con los mejores modelos encontrados. Esta opción fue inhabilitada para una comparación más directa del óptimo encontrado. En el caso de PILS, se descubrió que con tan solo 2 iteraciones del ILS era suficiente para resultados buenos. La función objetivo utilizada mide el BER y utiliza *Hold Out* con un 90% de los datos para el entrenamiento.

Como fue definido, el problema que se intenta resolver con este trabajo busca el mejor modelo posible en un tiempo razonable. Es por esto que se definieron dos métricas para comparar PILS con PSMS. En primer lugar, se desea evaluar la calidad del modelo encontrado por cada algoritmo. Para eso, cada algoritmo es entrenado conociendo únicamente la información contenida en el conjunto de entrenamiento, manteniendo separado el conjunto de validación. Luego, se entrena el modelo retornado usando todo el conjunto de entrenamiento y se prueba con el conjunto de validación, calculando el BER. La segunda métrica considerada es el tiempo que requirió cada algoritmo en su búsqueda.

Ambos algoritmos tienen componentes aleatorias que afectan significativamente su ejecución. Es por esto, que dos ejecuciones del mismo algoritmo con los mismos datos, no necesariamente retornan el mismo resultado, tanto en desempeño como en tiempo de ejecución. Es por esto, que se cada combinación de algoritmo y conjunto de datos fue ejecutada numerosas veces. Los resultados expuestos en este trabajo son el promedio de estas ejecuciones. El número de veces que se ejecutó cada algoritmo varía dependiendo del tiempo de ejecución que requería el mismo.

4.3.3. Resultados Experimento

Los resultados, expuestos en la **Tabla 5**, comparan el desempeño promedio y tiempo promedio requerido por PSMS y PILS. A partir de estos datos, es posible extraer dos observaciones relevantes. La primera observación, es que la diferencia entre los desempeños obtenidos por PILS y PSMS no es significativa. Si se considera que PSMS

es un algoritmo que ha demostrado ser capaz de encontrar modelos muy buenos, es posible que retorne el óptimo global (o un modelo de desempeño muy similar al óptimo global), por lo que sería imposible encontrar modelos con un desempeño significativamente mejor. La segunda observación, es que el tiempo que requieren para su búsqueda los dos algoritmos es muy diferente. En esta característica, PILS ha demostrado ser considerablemente más eficiente, con respecto al tiempo de ejecución, en su búsqueda. Este atributo es especialmente relevante para implementaciones prácticas de un algoritmo de este tipo.

Tabla 5: Comparación de desempeño del modelo encontrado y tiempo necesario para encontrarlo, entre PSMS y PILS, sobre distintos conjuntos de datos.

Conjunto de Datos	**PSMS**		**PILS**	
	BER	Tiempo (Minutos)	BER	Tiempo (Minutos)
ADA	0,192	193	0,185	20
GINA	0,035	922	0,042	220
HIVA	0,433	26394	0,378	718

Resulta fácil ver como el tiempo de ejecución crece considerablemente al aumentar el tamaño del conjunto de datos. Este es uno de los motivos por los que es tan relevante la construcción de un algoritmo que sea eficiente en cuanto a su uso del tiempo de ejecución. Por ejemplo, para el conjunto de datos HIVA, PILS requiere un promedio de 718 minutos (equivalente a un poco más de once horas), mientras que PSMS requiere un promedio de **26394** minutos (equivalente a un poco más de dos semanas y media). Además, en este caso particular, PILS consigue un desempeño considerablemente mejor que PSMS. Es clara la ventaja que tiene, para un problema de esas características, la implementación de un algoritmo como el aquí propuesto.

5. CONCLUSIONES

Esta investigación se desarrolló con el objetivo de construir e implementar un algoritmo que permita seleccionar el mejor modelo posible para un problema de aprendizaje supervisado, en un tiempo razonable. Para esto, se diseñó un algoritmo con un enfoque probabilístico, que permite hacer la búsqueda de modo eficiente en cuanto al tiempo de ejecución. En especial a lo largo de la experimentación, fue evidente la necesidad de un algoritmo que cumpla este propósito, ya que, el tiempo que requieren las metodologías actuales hace que cualquier implementación práctica de éstas sea inviable.

En el capítulo anterior fue posible ejemplificar la enorme necesidad de un algoritmo que permita resolver este problema en un tiempo razonable. Ya que, aun algoritmos del estado del arte en esta área, requieren de demasiado tiempo como para una aplicación práctica de los mismos. El algoritmo propuesto en este trabajo no solo logró resolver el problema con un desempeño comparable al de algoritmos disponibles hoy en día para esta tarea, sino que lo hizo utilizando entre el 5% y el 20% del tiempo requerido por los mismos.

Con los resultados publicados en este texto, resulta claro que los objetivos planteados fueron alcanzados de manera exitosa. Se logró retornar modelos de desempeño comparable con obtenidos por metodologías de vanguardia, en un tiempo considerablemente menor. Esto no solo validó las modificaciones hechas al algoritmo considerado como base, sino que mostró que este nuevo enfoque a la resolución de este tipo de problema es tanto eficaz como eficiente en su respuesta.

A lo largo de este trabajo se desarrolló un algoritmo de optimización probabilístico, con el objetivo de resolver, de modo eficiente, el problema de selección de modelo. Pero, el algoritmo desarrollado fue diseñado de modo genérico. Esto permitiría su utilización en otros contextos en donde se logren satisfacer algunas condiciones. Las únicas condiciones necesarias para la utilización de este algoritmo

como algoritmo de optimización, es que sea posible definirlo como un problema de programación lineal con una función objetivo estimada, que cumpla con los requisitos planteados en el diseño. Ver el efecto que tiene la aplicación de esta metodología a otros contextos escapa de los objetivo de este trabajo, por lo que únicamente se sugiere como un posible trabajo futuro en esta línea de investigación.

Como trabajo futuro, se pueden ver principalmente dos líneas de investigación. La primera línea de investigación, es continuar en la búsqueda de un mejor algoritmo de selección de modelo. En esta línea, podrían analizase nuevas opciones de función objetivo, contrastando el sacrificio en desempeño y la ganancia en tiempo de ejecución que producen. También, se podrían analizar modificaciones a la trayectoria de búsqueda, la cual no fue alterada durante este trabajo. Por último, se debería comparar el desempeño de PILS con otros algoritmos de selección de modelo disponibles, para una mayor validación de su calidad. La segunda linea de investigación es la aplicación de este algoritmo de optimización a otros contextos. Como fue mencionado en el párrafo anterior, es posible aplicar PILS a cualquier problema de búsqueda, siempre que se puedan cumplir con los supuestos del mismo.

6. REFERENCIAS

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* New York, USA: Springer.

Cortazar, E., & Mery, D. (2011). A Probabilistic Iterative Local Search Algorithm, Applied to Full Model Selection. *Progress in Pattern Recognition, Image Analysis, Computer Vision and Applications* (págs. 675-682). Pucon, Chile: Lecture Notes on Computer Science.

Devijver, P. A., & Kittler, J. (1982). *Pattern Recognition: A Statistical Approach.* Prentice-Hall International.

Escalante, H. J., Montes, M., & Sucar, L. E. (2009). Particle Swarm Model Selection. *J. Mach. Learn. Res.* , 405-440.

Glover, F., & Laguna, M. (1998). *Tabu Search.* Kluwer Academic Publishers.

Gorissen, D., De Tommasi, L., Croon, J., & Dhaene. (2008). Automatic model type selection with heterogeneous evolution: An application to RF circuit block modeling. *Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, (págs. 989-996). Hong Kong.

Guyon, I., Alamdari, A., Dror, G., & Buhmann, J. (2006). Performance Prediction Challenge . *International Joint Conference on Neural Networks 2006*, (págs. 1649-1656). Vancouver, BC.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stutzle, T. (2009). ParamILS: an automatic algorithm configuration framework. *J. Artif. Int. Res.* , 267-306.

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings IEEE International Conference on Neural Networks*, (págs. 1942-1948). Perth, Australia.

Kim, J.-H. (2009). Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics and Data Analysis* , 3735-3745.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *International Joint Conference on Artifcial Intelligence (IJCAI).*

Mery, D., Lillo, I., Loebel, H., Riffo, V., Soto, A., Cipriano, A., y otros. (2011). Automated Fish Bone Detection using X-ray Testing. *Journal of Food Engineering* , 485-492.

Polikar, R. (2006). Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE* , 21-45.

Smit, S. K., & Eiben, A. E. (2010). Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. *Aplications of Evolutionary Computation* , 542-551.

Wolpert, D. H., & Macready, W. G. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* , 67-82.