UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
VERITATE SOLA NOVIS INPONETUR VIRILIS TOGA. 1948

Facultad Matemática, Física y Computación
Departamento de Ciencia de la Computación

Trabajo en opción al título académico de
Máster en Ciencias de la Computación

Título

ESTUDIO COMPARATIVO DEL COMPORTAMIENTO DE MÉTODOS DE ESTIMACIÓN DE FUNCIONES CONTINUAS EN EL PRONÓSTICO DE LA TENSIÓN DE RUPTURA POR TERMOFLUENCIA EN ACEROS FERRÍTICOS.

Autor:
Ing. Carlos Alberto Donís Díaz

Tutores:
Dr. Carlos Morell Pérez
Dr. Eduardo Valencia Morales

Santa Clara, 2008

a Mely, Jenni y Yitsy

Agradecimientos

Agradezco a todas las personas que de una u otra forma han contribuido al desarrollo y culminación del presente trabajo. En especial:

- *A los Drs. Carlos Morell Pérez y Eduardo Valencia Morales por sus consejos y guía como tutores de la investigación,*
- *Al Dr. Ricardo Grau por su amable atención y colaboración precisa en el análisis estadístico,*
- *Al Dr. Rafael Bello por las ideas aportadas,*
- *A mis compañeros del Laboratorio de Inteligencia Artificial por su colaboración,*
- *A mi hermano, madre y padre por su ayuda constante,*
- *A Yitsy por constituir mi "grupo" de apoyo e imagen de futuro,*
- *A Jenni y ahora Mely, mis motivaciones e inspiraciones mayores.*

RESUMEN

En el presente trabajo se realiza un estudio del comportamiento de varios modelos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por termofluencia en aceros ferríticos, cuestión fundamental para el diseño de nuevas aleaciones dentro de la Ciencia de los Materiales. El estudio tiene como antecedentes el empleo, en esta problemática, de métodos empíricos, semi-empíricos y más recientemente de Redes Neuronales.

En la investigación se experimenta con varios modelos, desde aquellos tradicionales como son las Regresiones Lineales pasando por modelos bien establecidos como las Redes Neuronales (se utilizó un MLP) hasta modelos de reciente creación y desarrollo dentro del campo del aprendizaje automatizado como es el caso de las Máquinas de Soporte de Vectores para Regresión (SVMR) y los Procesos Gaussianos. También se utilizaron variantes del algoritmo de Optimización basado en Enjambre de Partículas (PSO) dentro de varios procesos de optimización implementados en la investigación.

Al analizar los resultados de los experimentos se pudo observar que los modelos de SVMR y Procesos Gaussianos resultaron los de mejor prestación. En la explicación teórica se justifica el mejor resultado de SVMR respecto al MLP debido al mejor comportamiento que ofrece el Principio de Minimización del Riesgo Estructural en que se basa el primer modelo respecto al Principio de Minimización del Riesgo Empírico en que se basa el segundo. Los Procesos Gaussianos, además de su eficiente comportamiento, permiten establecer el nivel de incertidumbre de la predicción en nuevos casos por lo cual se recomienda su uso en esta problemática.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Durante más de medio siglo, en muchas aplicaciones que abarcan campos tan disímiles como la industria petroquímica, energética y aeronáutica, se han utilizado extensivamente aceros ferríticos resistentes a la termofluencia. Lo anterior se debe a la excelente fiabilidad que estos aceros muestran para condiciones muy agresivas y un uso prolongado que incluye períodos de servicio como por ejemplo, 30 años.

En la historia del diseño de los aceros se puede observar un desarrollo progresivo, particularmente en el sentido de la creación de aceros que soportan altas temperaturas de vapor al utilizarse en plantas de energía contribuyendo a que éstas operen con gran eficiencia, o que son utilizados en turbinas de aviones garantizando un gran nivel de seguridad. Esta idea permite concluir que los principios básicos del diseño de aceros resistentes a la termofluencia, están bien establecidos y bien fundamentados en base a la experiencia.

La termofluencia, conocida dentro de la Ciencia de los Materiales por su vocablo en inglés como ***creep***, refiere un fenómeno físico que describe las deformaciones que ocurren en los metales a partir de transformaciones internas provocadas por el sometimiento de los mismos a altas temperaturas, durante largos períodos de servicio. Así mismo, la ***tensión de ruptura por creep***, se puede describir como el valor de tensión a partir del cual el metal no es capaz de recobrar su estructura original influenciado por las condiciones antes mencionadas, definiendo el pase de la fase de elasticidad de dicho metal a una fase de plasticidad. Este último concepto es uno de los pilares fundamentales utilizados en el diseño de nuevas aleaciones que se crean para ser utilizadas en condiciones como las descritas y constituye una de las fuentes de motivación de la presente investigación.

Los aceros deben tener una microestructura estable que contenga finos carburos aleados resistentes a los movimientos de las dislocaciones; sin embargo, es inevitable que durante largos periodos de servicio o condiciones muy críticas, se produzcan cambios. Es necesario, por tanto, garantizar un potencial de endurecimiento por solución sólida suficiente que permita mantener de forma adecuada la microestructura y asegurar así una elevada resistencia a las deformaciones por *creep* a largo plazo. También se necesitan otros requerimientos como es la soldabilidad y resistencia a la corrosión y la oxidación.

Teniendo en cuenta estas ideas se puede deducir la gran cantidad de variables que intervienen en el diseño de los aceros para lograr determinadas cualidades y propiedades mecánicas y lo difícil que resulta poder expresar alguna relación cuantitativa o interacción que tenga lugar entre dichas variables.

En el desarrollo de los aceros, se han utilizado varios métodos empíricos para estimar la tensión de ruptura por *creep* destacándose recientemente, el empleo de redes neuronales. La utilización de estos métodos ha sido de manera espontánea sin tener en cuenta otros métodos de desarrollo reciente y que pudieran tener resultados satisfactorios. Esta idea puede ser constatada a partir del análisis de bibliografía actualizada expuesto en el *Capítulo 1* de este informe.

Los aspectos descritos conducen al planteamiento del **problema científico** de la presente investigación. El mismo está relacionado con el hecho de que existen varios métodos de estimación de funciones continuas que no han sido utilizados en el pronóstico de la tensión de ruptura por *creep* y por tanto no existe un criterio explícito que fundamente que los resultados que se obtienen de los métodos utilizados hasta la actualidad, sean los más eficientes.

La presente investigación plantea un desarrollo en función de objetivos que permiten dar solución al problema científico planteado anteriormente.

Hipótesis. En el pronóstico de la Tensión de Ruptura por *Creep* en aceros ferríticos, el empleo de métodos de estimación de funciones continuas de reciente desarrollo dentro del campo del aprendizaje automatizado, permite obtener mejores resultados que el empleo de métodos tradicionales.

Objetivo General: Realizar un análisis comparativo del comportamiento de diferentes métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por *creep* en aceros ferríticos, de forma tal que se pueda consolidar una base teórico-práctica que fundamente el comportamiento de los mejores métodos.

Objetivos específicos:

1. Establecer un marco teórico referencial que permita describir el estado actual de:

- el empleo, en la Ciencia de los Materiales, de métodos empíricos para el pronóstico de la tensión de ruptura por *creep* en los aceros,
- las características de varios métodos novedosos de estimación de funciones continuas desarrollados dentro del campo de la inteligencia artificial y la estadística.

2. Implementar una plataforma de clases que permita utilizar el método de Optimización basado en Enjambre de Partículas (PSO por sus siglas en inglés de *Particle Swarm Optimization*) en diferentes roles para la estimación de funciones continuas dentro del ambiente de WEKA.
3. Desarrollar una base teórico-práctica que permita identificar y justificar el comportamiento de varios métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por *creep* en aceros ferríticos. Lo anterior se garantiza mediante:
   - *Dimensión práctica*: la implementación de experimentos con datos de *creep* de forma tal que se constituya una base de datos como fuente analítica.
   - *Dimensión teórica*: el establecimiento de los fundamentos de las diferencias que se obtengan en los resultados a partir de los diferentes métodos.

La novedad científica de la presente investigación está vinculada con sus objetivos y se refiere al establecimiento de un estudio descriptivo que fundamente el comportamiento de varios métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por *creep* en aceros ferríticos.

A la investigación se le pueden atribuir los siguientes valores:

Teórico: brinda un marco referencial que fundamenta el comportamiento del empleo de varios métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por *creep*.

Metodológico: crea las bases para el empleo de métodos de estimación de funciones continuas nunca antes utilizados en el pronóstico de la tensión de ruptura por *creep*. Además, establece una forma de crear nuevos clasificadores dentro de WEKA mediante la utilización de PSO.

Valor práctico: la investigación permite:

- contar con un nuevo clasificador dentro de WEKA y además una plataforma de objetos que permite la utilización de PSO en diferentes roles dentro de WEKA,
- establece los preceptos para un software que permita pronosticar la tensión de ruptura por *creep*.

Valor social: La tensión de ruptura por *creep* es uno de los indicadores fundamentales en el diseño de nuevas aleaciones que se utilizan específicamente en industrias como la energética, petroquímica, aeronáutica, etc. La presente investigación al proveer de un fundamento teórico y una base para la implementación de una herramienta práctica con este fin, produce un impacto directo en dichas industrias con el consiguiente beneficio social que trasmite a la sociedad.

Para la presentación del contenido y resultados de la investigación se utiliza el presente informe de Tesis de Maestría para cuyo objetivo se estructura de la siguiente forma:

- un primer capítulo donde se establece un marco teórico referencial desarrollado a partir de la consulta de bibliografía actualizada sobre los temas que constituyen la plataforma de la investigación,
- un segundo capítulo que expone los experimentos realizados, así como la implementación de los aspectos necesarios para desarrollar una comparación de resultados entre los diferentes métodos de estimación de funciones continuas utilizados,
- un tercer capítulo que constituye el análisis y fundamentación de los resultados obtenidos,
- un acápite de conclusiones y uno de recomendaciones donde se exponen la forma en que se lograron los objetivos propuestos, así como las pautas a tener en cuenta para futuras investigaciones y
- finalmente un acápite de referencias bibliográficas que relaciona los documentos consultados para el desarrollo de la presente investigación.

CAPÍTULO 1.

Métodos de estimación de funciones continuas. Antecedentes en el pronóstico de la tensión de ruptura por creep.

1.1 Antecedentes del empleo de métodos empíricos en el pronóstico de la tensión de ruptura por *creep.*

El diseño de materiales involucra la optimización simultánea de un gran número de parámetros, incluso sin tener bien definidas las interacciones existentes entre los mismos. La "modelación" puede contribuir a la creación de una nueva teoría capaz de lidiar con una adecuada complejidad pretendiendo, como objetivos prácticos, acelerar el proceso de diseño y minimizar el uso de recursos. En el contexto del diseño de aceros ferríticos resistentes al *creep* se han realizado varios trabajos teniendo como núcleo, la modelación de la tensión de ruptura por *creep*.

H. K. D. H. Bhadeshia y *T. Sourmail* [1] realizaron un análisis detallado de los éxitos y fallas de varios modelos propuestos en este campo los cuales han sido revisados recientemente en la literatura [2, 3].

Desde el punto de vista de la ciencia ordinaria, cuyos preceptos parten de intentar reducir el problema hasta que pueda ser descrito rigurosamente utilizando principios fundamentados en conocimientos contemporáneos, la deformación por *creep* en los metales se ha expresado en función de la tensión aplicada (σ), la temperatura absoluta (T) y el tamaño del grano (d) según:

$$\varepsilon_{SS} = a_1 \left(\frac{DGb}{kT} \right) \left(\frac{b}{d} \right)^m \left(\frac{\sigma}{G} \right)^n$$

Donde ε_{SS} es la deformación por creep en el estado estacionario, a_1 es usualmente una constante empírica, D es un coeficiente de difusión apropiado, b es la magnitud del vector de Burgers y G es el módulo de cizalladura. El exponente de tensión n y el exponente del tamaño del grano m son dependientes del mecanismo de *creep*. Desafortunadamente esta aproximación de leyes de potencia nunca ha tenido una adecuada descripción en materiales prácticos; es decir, no tiene en cuenta muchos tipos de defectos, solutos, microestructuras que dependen del tiempo, etc que son la base de los aceros comerciales.

Partiendo de las fallas descritas se han desarrollado numerosas aproximaciones que sí pueden lidiar con niveles de complejidad adecuados. Entre ellos se destaca un gran número de métodos empíricos o semi-empíricos que pueden obtener una representación exacta de los datos experimentales y al mismo tiempo permiten la extrapolación de los datos de *creep* con diferentes niveles de éxito. Descripciones de estos métodos pueden encontrarse en la literatura [2-24].

Un método popular [25] involucra la ecuación *θ*-parametrizada:

$$\underbrace{\varepsilon = \theta_1\left[1-\exp\{-\theta_2 t\}\right]}_{\text{Vel. def. prim.}} + \underbrace{\theta_3\left[\exp\{\theta_4 t\}-1\right]}_{\text{Regimen Acelerac.}}$$

donde los θ_i son obtenidos mediante ajustes a partir de los datos experimentales y t es el tiempo a una temperatura dada. El primer componente describe la velocidad de deformación primaria o decadente y el segundo, el régimen de aceleración.

Otro modelo es el siguiente. La teoría de estado estacionario de *creep* conduce a una ecuación de la forma:

$$\dot{\varepsilon} = a_3\sigma^n \exp\left\{-\frac{Q}{RT}\right\}$$

donde a_3 es una constante empírica. Esta ecuación puede ser integrada para encontrar el tiempo de ruptura por *creep* (t_r) como:

$$\ln\{t_r\} = \ln\left\{\frac{\varepsilon_r}{a_3}\right\} - n\ln\{\sigma\} + \frac{Q}{RT}$$

No obstante, en la práctica se ha visto [26, 27,28] que se pueden obtener ajustes ligeramente mejores utilizando

$$\ln\{t_r\} = a_4 + a_5\sigma + a_6 T$$

donde las a_i son constantes ajustadas.

A pesar de que tanto estos como otros modelos parámetricos, han sido ampliamente utilizados, queda claro que no son lo suficientemente generales como para lidiar con un

gran número de variables. Por este motivo, se ha propuesto recientemente [27], el empleo de las redes neuronales para la solución de la problemática del *creep*.

En este sentido se han desarrollado varios trabajos basados en el modelo de red neuronal conocido como *Multi-layer Perceptron* (MLP) el cual ha sido entrenado en un ambiente Bayesiano desarrollado por *D. Mackay* [26] y ha demostrado ser superior en la extrapolación y representación de datos de *creep* [1]. Por ejemplo *T. Sourmail* [27] creó un modelo basado en esta técnica para estimar la tensión de ruptura por *creep* en aceros austeníticos inoxidables en función de la composición química (16 elementos diferentes), el tratamiento térmico y la temperatura y tiempo de servicio. Otros trabajos similares han utilizado el modelo para aceros ferríticos como los de *F. Brun* y colectivo de autores [28] y *D. Cole* y colectivo de autores [29].

Debido a los resultados obtenidos, se ha llegado a sugerir [1] que este método de estimación no lineal mediante redes neuronales entrenadas en un ambiente bayesiano puede resultar una vía mejor para el tratamiento de datos de *creep* en el desarrollo de productos y diseño de estándares, si se compara con las técnicas que se utilizan en la actualidad por parte de, por ejemplo, el ECCC (*European Creep Collaborative Committee*).

1.1.1 Redes neuronales en el pronóstico de la tensión de ruptura por creep.

En este epígrafe se describe el modelo de red neuronal utilizado en los trabajos referenciados para estimar la tensión de ruptura por *creep*.

En el análisis de regresión los datos se adaptan a una relación específica que es usualmente lineal. En estos casos el resultado es una ecuación que determina la suma pesada de las entradas x_j. Si se denotan los pesos como w_j se puede plantear la ecuación que estima las salidas como:

$$y = \sum_j w_j x_j + \theta$$

Una red neuronal es un método de regresión mucho más general que permite adaptar una función no lineal a los datos experimentales, de forma muy flexible y caracterizada por una topología determinada.

El modelo de red neuronal utilizado por *T.Sourmail* presenta una topología compuesta por nodos o neuronas organizadas en capas, existiendo una capa de entrada, una capa oculta y una capa de salida. Entre los nodos de las diferentes capas existen conexiones que establecen relaciones entre ellos y tienen asociado un determinado peso que representa la fortaleza de la misma. Cada nodo oculto (perteneciente a la capa oculta) recibe como entrada la suma pesada de los valores de salida de los nodos cuyas conexiones convergen en él y obtiene como salida el resultado de la evaluación de una función matemática cuyos parámetros constituyen el valor de entrada al nodo y un bias o umbral (equivalente a las constantes en los análisis de regresión lineal) específico del propio nodo. Las salidas de cada nodo de la capa oculta constituyen los parámetros de entrada del nodo de la capa de salida cuya función es obtener la suma pesada de todos esos valores, lo cual constituye la salida de la red. Los nodos de la capa de entrada solo "trasmiten" los valores de las variables de entrada de la red hacia los nodos de la capa oculta.

La flexibilidad de la función que representa la red neuronal está relacionada con el número de nodos ocultos ***i***. Así, en los modelos referenciados anteriormente, la variable dependiente *y* obtenida como el resultado del nodo de la capa de salida se expresa como:

$$y = \sum_i w_i h_i + \theta \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

donde w_i es el peso de la conexión desde el *i-ésimo* nodo de la capa oculta hasta el nodo de salida, θ es el bias del nodo de salida y

$$hi = \tanh\left(\sum_j w_{ij} x_j + \theta_i \right) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

donde x_j constituyen las ***j*** variables que provienen de la capa de entrada y de las cuales depende la salida ***y***, w_{ij} son los pesos de las conexiones desde cada nodo de entrada ***j*** hasta cada nodo ***i*** de la capa oculta y θ_i es el bias del nodo ***i***. Como función dentro del nodo, se utiliza la tangente hiperbólica (*tanh*). La combinación de la ecuación (2) con un conjunto de pesos w_{ij}, biases, valor de ***i*** y valores máximos y mínimos de las variables de entrada conforma la red neuronal.

Una topología de red neuronal como la descrita se conoce como MLP. En los trabajos referenciados se utilizó un MLP formado por 37 nodos en la capa de entrada correspondiendo con cada una de las variables que se describen en la ***Tabla 1*** del epígrafe 2.1 del presente trabajo, una capa oculta con un número variable de nodos y una capa de salida con un solo nodo cuya salida corresponde al estimado de la tensión de ruptura por *creep*. La estructura es similar a la mostrada en la ***Figura 1***.

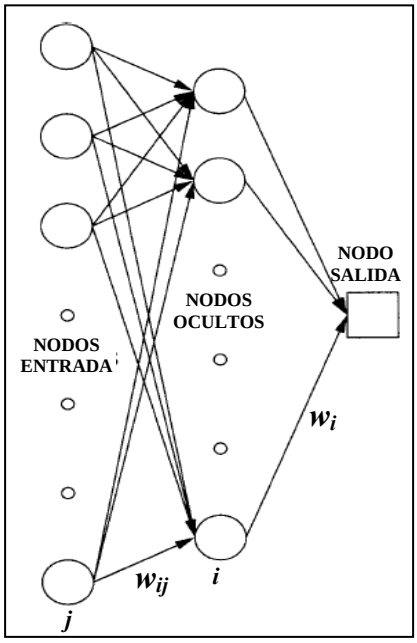


Figura 1. *Estructura de un MLP con una sola capa oculta*

El MLP, mediante un proceso denominado “de entrenamiento” y teniendo como base un conjunto de ejemplos de datos de entrada y salida, logra adaptar el conjunto de pesos y biases de manera que relaciona de forma adecuada y no lineal las entradas con la salida de la red. Una vez concluido este proceso, la estimación de valores de salidas para valores de entrada no “conocidos” por la red es muy rápida.

Como se ha visto, la disponibilidad de funciones complejas y flexibles no es tan restringida como en las regresiones lineales donde la forma de la ecuación se define explícitamente antes del análisis.

El MLP puede captar interacciones entre las entradas debido a que los nodos ocultos son no lineales. La naturaleza de esas interacciones está implícita en los valores de los pesos, aunque no siempre son fáciles de interpretar. Por ejemplo, pueden existir más de un par

de interacciones en cuyo caso el problema se convierte más difícil de visualizar con solo examinar los pesos. Por tanto un método mejor es utilizar la red para hacer predicciones y ver como éstas se relacionan con varias combinaciones de las entradas.

D. MacKay ha desarrollado un tratamiento particular de las redes neuronales en un ambiente Bayesiano [26] permitiendo el cálculo de barras de error que representan la incertidumbre en el ajuste de los parámetros. El método reconoce que existen varias funciones que pueden ser ajustadas o extrapoladas en regiones de incertidumbre del espacio de entrada sin comprometer indebidamente el ajuste en regiones adyacentes que sí contienen cantidades adecuadas de datos precisos. En vez de calcular un único conjunto de pesos, se utiliza una distribución de probabilidades de pesos lo cual permite definir la incertidumbre del ajuste. Estas barras de error son más largas cuando los datos están dispersos o son localmente ruidosos.

1.2 Métodos de estimación de funciones continuas.

El desarrollo de métodos empíricos de estimación de funciones continuas se ha caracterizado por su gran consistencia (existen métodos convencionales bien fundamentados y probados) y evolución dinámica. Estos aspectos han sido asumidos por campos como el aprendizaje estadístico y el aprendizaje automatizado existiendo una tendencia a la vinculación de modelos propuestos desde ambas perspectivas, así como a la creación de espacios de trabajo unificados para estas técnicas.

El ambiente de trabajo para el análisis de conocimientos WEKA (por sus siglas en inglés *Waikato Environment for Knowledge Analysis*), desarrollado en la Universidad de Waikato, Nueva Zelanda posee una interesante y actualizada plataforma de modelos que se utilizan en la estimación de funciones continuas y que han sido desarrollados dentro del campo de la estadística y de la inteligencia artificial. En el presente trabajo se emplearon varios de estos métodos entre los que destacan algunos clásicos y bien establecidos como la regresión lineal y otros de reciente creación y desarrollo como las Máquinas de Vectores de Soporte y los Procesos Gaussianos. Igualmente se utilizó un modelo enmarcado en la categoría de "*Modelos Bioinspirados*" denominado PSO por sus siglas en inglés de *Particle Swarm Optimization* (Optimización mediante enjambre de partículas).

Como resultado del análisis de bibliografía y literatura actualizada, en los siguientes acápites se establece un marco teórico referencial con los preceptos de los métodos fundamentales y más novedosos empleados en el presente trabajo.

1.2.1 El Perceptron Multicapa o Multilayer Perceptron (MLP).

Como fue referenciado anteriormente, el MLP es un modelo de red neuronal con una topología compuesta por tres tipos de capas: de entrada, ocultas y de salidas. Esta arquitectura suele entrenarse mediante el algoritmo denominado retropropagación de errores o *BackPropagation*. En el epígrafe se expone una descripción más general del modelo y de su método clásico de entrenamiento.

La estructura del MLP con solo una capa oculta puede ser representada como una generalización de la ***Figura 1*** tal y como se muestra a continuación.

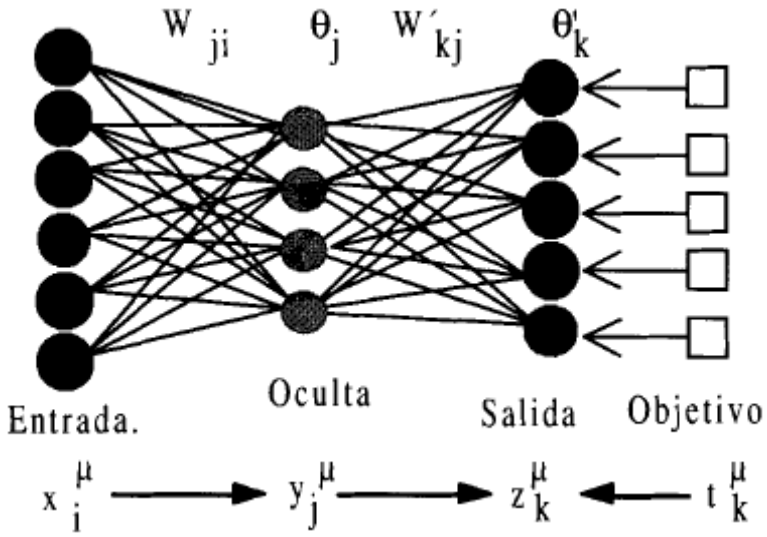


Figura 2. *Estructura de un MLP con una capa oculta y k nodos en la capa de salida*

Denominaremos x_i a las entradas de la red, y_j a las salidas de la capa oculta y z_k a las de la capa final (y globales de la red); t_k serán las salidas objetivo (*target*). Por otro lado, w_{ij} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales. La operación de un MLP con una capa oculta y neuronas de salida lineal (estructura que constituye un aproximador universal de funciones [30]) se expresa matemáticamente de la siguiente manera

$$Z_k = \sum_j w'_{kj} y_j - \theta'_k = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i - \theta_j \right) - \theta'_k \quad (3)$$

siendo $f(.)$ de tipo sigmoideo, como por ejemplo, las siguientes

$$f(x) = \frac{1}{1+e^{-x}} \qquad f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$$

proporcionando la primera una salida en el intervalo [0, +1], y en el [-1, +1] la segunda.

Ésta es la arquitectura más común de un MLP, aunque existen numerosas variantes, como incluir neuronas no lineales en la capa de salida (del mismo tipo que las sigmoideas descritas anteriormente, solución que se adopta especialmente en problemas de clasificación), introducir más capas ocultas, emplear otras funciones de activación, limitar el número de conexiones entre una neurona y las de la capa siguiente, introducir dependencias temporales o arquitecturas recurrentes.

1.2.1.1 Aprendizaje por retropropagación de errores (*BackPropagation*).

En [30] se puede obtener una descripción detallada de éste método. A los efectos de la presente descripción, se utilizará un modelo de MLP como el de la ***Figura 2***; se considerará entonces un modelo con ***i*** nodos en la capa de entrada, ***j*** nodos en la capa oculta y ***k*** nodos en la capa de salida. Se supone además que se cuenta con ejemplos de entrada $\boldsymbol{x^{\mu}}$ (μ = 1,…, m). La operación global del MLP se expresa como:

$$Z_k^{\mu} = \sum w'_{kj} y_j^{\mu} - \theta'_k = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i^{\mu} - \theta_j \right) - \theta'_k$$

La función costo de la que se parte es el error cuadrático medio

$$E\left(w_{ji}, \theta_j, w'_{kj}, \theta'_k\right) = \frac{1}{2} \sum_{\mu} \sum_k \left[t_k^{\mu} - f\left(\sum_j w'_{kj} y_j^{\mu} - \theta'_k \right) \right]^2$$

El método de entrenamiento *Backpropagation* lleva a cabo la minimización del error mediante descenso por el gradiente, existiendo un gradiente respecto a los pesos de la capa de salida y otro respecto a los de la oculta

$$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \qquad \delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}}$$

Las expresiones de actualización de los pesos se obtienen sólo con derivar, teniendo en cuenta las dependencias funcionales y aplicando adecuadamente la regla de la cadena

$$\delta w'_{kj} = \varepsilon \sum_{\mu} \Delta'^{\mu}_{k} y^{\mu}_{j} \text{, con } \Delta'^{\mu}_{k} = \left[t^{\mu}_{k} - f\left(v'^{\mu}_{k} \right) \right] \frac{\partial f\left(v'^{\mu}_{k} \right)}{\partial v'^{\mu}_{k}}$$

$$\delta w_{ji} = \varepsilon \sum_{\mu} \Delta^{\mu}_{j} x^{\mu}_{i} \text{, con } \Delta^{\mu}_{j} = \left(\sum_{k} \Delta'^{\mu}_{k} w'_{kj} \right) \frac{\partial f\left(v^{\mu}_{j} \right)}{\partial v^{\mu}_{j}}$$

La actualización de los umbrales (*bias*) se realiza haciendo uso de estas mismas expresiones, considerando que el umbral es un caso particular de peso, cuya entrada es una constante igual a -1.

En estas expresiones está implícito el concepto de propagación hacia atrás de los errores que da nombre al algoritmo. En primer lugar se calcula la expresión Δ'^{μ}_{k} que se denomina señal de error, por ser proporcional al error de la salida actual de la red, con el que calculamos la actualización $\delta w'_{kj}$ de los pesos de la capa de salida. A continuación se propagan hacia atrás los errores Δ'^{μ}_{k} a través de las conexiones, proporcionando así las señales de error Δ^{μ}_{j}, correspondientes a las conexiones de la capa oculta; con éstas se calcula la actualización δw_{ji} de las conexiones ocultas. El algoritmo puede extenderse fácilmente a arquitecturas con más de una capa oculta siguiendo el mismo esquema.

Los pesos iniciales se seleccionan de manera aleatoria. Normalmente son números pequeños, positivos y negativos.

En el entrenamiento, se lleva a cabo una fase de ejecución para todos y cada uno de los ejemplos del conjunto de entrenamiento, se calcula la variación en los pesos debido a cada ejemplo, se acumulan, y solamente entonces se procede a la actualización de los pesos. A esta variante se le conoce como aprendizaje por lotes. Una variación común al algoritmo consiste en actualizar los pesos de las conexiones tras la presentación de cada ejemplo, este esquema se conoce como aprendizaje en serie y es habitualmente utilizado

en los problemas donde se dispone de un numeroso conjunto de ejemplos de entrenamiento.

A la idea original del algoritmo se le han realizado modificaciones para resolver los problemas de la lenta convergencia o incrementar la capacidad de generalización del aprendizaje. Para el caso de la lenta convergencia los propios inventores propusieron incluir en el algoritmo un término denominado momento (*momentum*), consistente en añadir al cálculo de la variación de los pesos un término adicional proporcional al incremento de la iteración anterior (inercia), quedando:

$$\delta w'_{kj}(t+1) = -\varepsilon \frac{\partial E}{\partial w'_{kj}}\Big|_t + \alpha \delta w'_{kj}(t-1) \qquad \delta w_{ji}(t+1) = -\varepsilon \frac{\partial E}{\partial w_{ji}}\Big|_t + \alpha \delta w_{ji}(t-1)$$

con α un parámetro entre 0 y 1, que se suele tomar próximo a 1 (≈ 0.9). De esta manera, si los incrementos en un determinado peso tienen siempre el mismo signo, las actualizaciones en cada iteración serán mayores; sin embargo, si los incrementos en cierto peso oscilan (a veces son positivos, otras negativos), el incremento efectivo (acumulado) se reduce al cancelarse. Así, en zonas estrechas y profundas de la hypersuperficie de error (con forma de valle angosto), los pesos correspondientes a la dimensión estrecha (que sin el término de momento oscilarían de un lado al otro del valle) sufren incrementos pequeños, mientras que los de las direcciones que descienden directamente al fondo se ven potenciados. Ésta es una manera de aumentar el ritmo de aprendizaje efectivo en determinadas direcciones.

1.2.2 Redes de funciones de base radial.

El modelo de redes neuronales de funciones de base radial o RBF (*Radial Basis Functions*) es un modelo de red unidireccional para aproximación funcional, considerado de tipo hibrido por incorporar aprendizaje supervisado y no supervisado. Al igual que el MLP, las RBF permiten modelar con relativa facilidad sistemas no lineales arbitrarios, con la particularidad de que el tiempo requerido para su entrenamiento suele ser mucho más reducido que el del *BackPropagation* clásico.

La arquitectura de una red RBF cuenta con tres capas de neuronas: de entradas, oculta y de salida (similar a un MLP de una sola capa oculta). Las neuronas de entrada, como

suele ser habitual, simplemente envían la información del exterior hacia las neuronas de la capa oculta. Las neuronas de la capa de salida son lineales, esencialmente calculan la suma ponderada de las salidas que proporciona la capa oculta.

La diferencia fundamental entre la arquitectura de este modelo y la del MLP se centra en la operación de las neuronas ocultas. Éstas, en vez de computar la suma ponderada de las entradas y aplicarle una función de tipo sigmoideo, operan en base a la distancia que separa el vector de entradas respecto del vector que cada una almacena (denominado centroide), cantidad a la que aplican una función radial con forma gaussiana (***Figura 3***). Es decir, así como en el MLP las neuronas ocultas poseen una respuesta de rango infinito (cualquier vector de entrada, con independencia del lugar del espacio de entrada de donde proceda puede causar que la neurona se active), en el RBF las neuronas son de respuesta localizada, pues solo responden con una intensidad apreciable cuando el vector de entradas presentado y el centroide de la neurona pertenecen a una zona próxima en el espacio de las entradas.

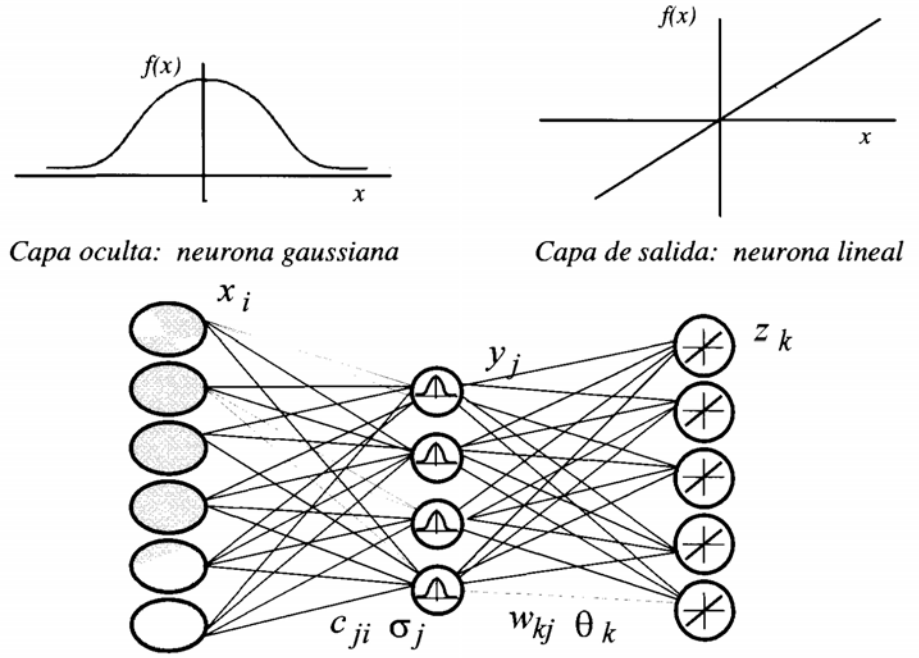


Figura 3. *Arquitectura del RBF y forma de las funciones de activación*

Descripción matemática del modelo.

Denominaremos x_i a las entradas de la red, y_j serán las salidas de la capa oculta, y z_k las salidas de la capa final (y globales de la red). Cada neurona ***j*** de la capa oculta almacena

un vector c_{ji}, el centroide; como en la red de Kohonen cada una de estas neuronas calcula la distancia euclidiana r_j que separa el vector de entradas x_i de su centroide

$$r_j^2 = \left\| \mathbf{x} - \mathbf{c}_j \right\|^2 = \sum_i \left(x_i - c_{ji} \right)^2$$

La salida de la neurona y_j se calcula a partir de una función de activación denominada función radial $\phi(r)$. Una de las más típicas es la función gaussiana

$$\phi(r) = e^{-r^2/2\sigma^2} \tag{4}$$

En ocasiones se emplean funciones diferentes, aunque de similar dependencia radial, como la siguiente

$$\phi(r) = r^2 \ln(r)$$

El término ***función de base radial*** procede precisamente de la simetría radial de éstas funciones (el nodo da una salida idéntica para aquellos patrones que distan lo mismo del centroide). En lo adelante se utilizará, a los efectos de esta descripción, la función gaussiana (4). El parámetro de normalización σ (o factor de escala) mide la anchura de la gaussiana, y equivaldría al radio de influencia de la neurona en el espacio de las entradas; a mayor σ la región que la neurona *domina* en torno al centroide es más amplia (***Figura 4***).

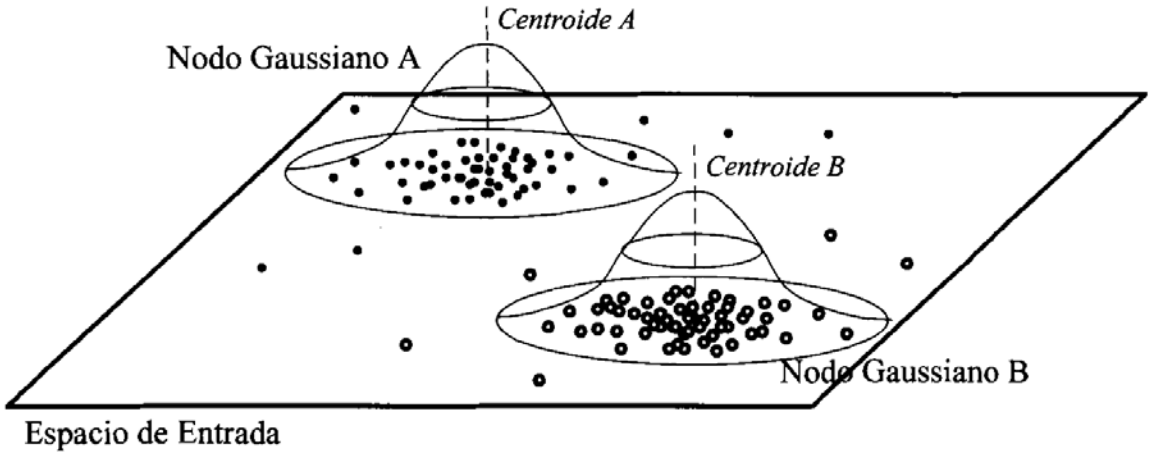


Figura 4. *Respuesta localizada de las neuronas ocultas en el RBF (nodos gaussianos). Los puntos representan patrones en el espacio de las entradas, que en su mayoría se agrupan en torno a dos centros.*

Recopilando las expresiones introducidas, la salida de la neurona oculta ***j*** se escribirá

$$y_j = e^{-r_j^2/2\sigma_j^2} = e^{-\sum_i \left(x_i - c_{ji}\right)^2 / 2\sigma_j^2}$$

Así, si el vector de entradas coincide con el centroide de la neurona ***j*** ($x=c_j$), ésta responde con máxima salida (la unidad). Es decir, cuando el vector de entradas se sitúa en una región próxima al centroide de una neurona, ésta se activa, indicando que "reconoce" el patrón de entrada; si el patrón de entrada es muy diferente del centroide, la respuesta tiende a cero. Hemos considerado funciones $\phi(.)$ simétricas, en algunos casos se elige una σ diferente para cada dirección, σ_{ji}, adquiriendo formas elipsoidales.

Las salidas de las neuronas ocultas son a su vez las entradas de las neuronas de salida, las cuales calculan su respuesta z_k de la forma:

$$z_k = \sum_j w_{kj} y_j + \theta_k = \sum_j w_{kj} \phi(r_j) + \theta_k$$

siendo w_{kj} el peso que conecta la neurona oculta ***j*** con la de salida k, y θ_k un parámetro adicional de la neurona k, que por similitud con el *Perceptron* se denomina umbral (*bias*). Puede observarse que esta expresión es completamente similar a la ecuación (3), que define la operación de un MLP con una capa oculta y neuronas de salida lineales, sólo que en aquel caso la función de activación de las neuronas ocultas era de tipo sigmoideo. Se puede demostrar que la arquitectura RBF basada en funciones de respuesta localizada $\phi(.)$ (como las gaussianas), constituye un aproximador universal de funciones [30]. Cada nodo gaussiano se ocupa de una zona del espacio, y el conjunto de nodos debe cubrir totalmente la zona de interés (***Figura 5***). Este cubrimiento debe llevarse a cabo de la forma más suave posible, lo cual se controla con el número de nodos de la capa oculta y con la anchura σ.

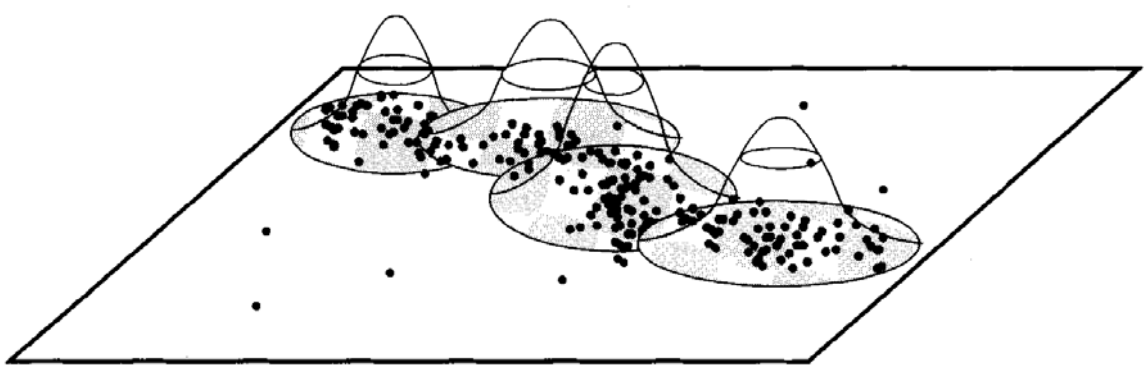


Figura 5. *Cuatro nodos gaussianos cubren el espacio de trabajo*

1.2.2.1 Aprendizaje en las RBF.

Existen distintas cuestiones a abordar en el trabajo con las RBF. La primera es cómo elegir el número de nodos radiales (ocultos). Cada nodo radial cubre una parte del espacio de entrada (***Figura 5***), de modo que habrá que elegir el número de nodos adecuados que cubra suficientemente (para una aplicación dada) dicho espacio. El problema surge al tratar con espacios de entrada de muchas variables, pues el número de nodos necesarios crece exponencialmente al hacerlo la dimensión. De este modo, debe llegarse a un compromiso entre el número de nodos radiales seleccionado, el error que se alcanza en el ajuste de los patrones de aprendizaje, y la capacidad de generalización (puede aparecer sobreajuste al tomar demasiadas neuronas ocultas).

Además de hacer uso del socorrido método de prueba y error, ampliamente empleado ya en el algoritmo *BackPropagation*, pueden utilizarse otros procedimientos más o menos automáticos para determinar el número de neuronas ocultas en una RBF. Por ejemplo, puede comenzarse con un determinado (reducido) número de nodos gaussianos, y si algún patrón de entrada no activa en suficiente medida ninguna de estas neuronas ocultas, se considera que es necesario introducir una nueva que de cuenta de la presencia de una nueva clase de patrones. Un procedimiento así se apunta en el modelo denominado algoritmo autoorganizado jerárquico (*Hierarchically Selforganizing Learning Algorithm*) de *S. Lee* [31].

Determinado de una u otra manera el número de nodos radiales, se trata a continuación, de encontrar los parámetros de la arquitectura. En principio podría tomarse una

aproximación global al problema, aplicando a la arquitectura particular de una red RBF el método de descenso por el gradiente, de similar forma al algoritmo *BackPropagation*.

No obstante, suele emplearse un aprendizaje por etapas, en el que en primer lugar se realiza el entrenamiento de las neuronas ocultas gaussianas, para finalmente proceder al entrenamiento de las neuronas de salida.

En el entrenamiento de los nodos gaussianos debe determinarse en primer lugar el valor de los centroides c_{ji}, pudiéndose aplicar para ello el conocido algoritmo de las k-medias (*k-means* [33]), o cualquier otro algoritmo no supervisado para agrupamiento (*clustering*). En el caso del algoritmo de las k-medias, k hace referencia al número de grupos (clusters) que se trata de encontrar, que en este caso se corresponderá con el número de nodos gaussianos propuesto de partida. Así, en este procedimiento hay que proponer en primer lugar un número k de grupos, es decir, de neuronas; a continuación se sigue el siguiente proceso:

1. Se eligen los valores de los k centroides c_j de partida. Suelen tomarse como centroides los primeros k patrones de aprendizaje (la elección concreta no es relevante para el resultado final).
2. En cada iteración t se reparten los patrones de aprendizaje x entre las k neuronas. Cada patrón se asigna a la neurona de cuyo centroide dista menos.
3. Se calculan los nuevos centroides de cada neurona como promedio de los patrones de aprendizaje asignados en el paso (2). Así, llamando N_j al número de patrones que han correspondido a la neurona j en el reparto, se tiene

$$c_{ji} = \frac{1}{N_j} \sum_{\mathbf{x} \in \text{neurona } j} \mathbf{x}$$

4. Si los valores de los centroides no han variado respecto de la iteración anterior, el algoritmo ya ha convergido. Si no es así, volver a (2)

Obtenidos los centroides, se procederá al cálculo de los parámetros de escala σ_j de cada neurona, para lo cual suele hacerse uso de criterios heurísticos. Un procedimiento se basa en calcular de forma aproximada el radio de influencia en el espacio de las entradas de cada neurona en relación a las demás, para lo cual se procede de la siguiente forma: para

calcular el σ_j, de la neurona j se seleccionan los centroides de las N neuronas que están más próximos al del nodo j, y a continuación se calcula el promedio de las distancias cuadráticas entre ellos, es decir

$$\sigma_j^2 = \frac{1}{N}\sum_{l=1}^{N}\left\|\mathbf{c}_l - \mathbf{c}_j\right\|^2 = \frac{1}{N}\sum_{l=1}^{N}\sum_{k}\left(c_{lk} - c_{ik}\right)^2$$

El caso extremo $N = 1$, que consiste en tener en cuenta únicamente el nodo más cercano, obviamente resulta el más rápido de calcular y sin embargo, proporciona buenos resultados en muchos casos [34].

Otro procedimiento es el indicado en [35], consistente en el cálculo del promedio de la distancia de diversos patrones representativos al centroide

$$\sigma_j^2 = \frac{1}{N_j}\sum_{\mathbf{x}\in \text{nodo } j}^{p}\left\|\mathbf{x} - \mathbf{c}_j\right\|^2$$

donde se ha denominado N, al número de patrones tomados para el cálculo del factor de escala del nodo j. Calculados los factores de escala de una forma u otra, con ello finaliza el entrenamiento de las neuronas de la capa oculta.

Por último, se procede al entrenamiento de las neuronas de salida. Haciendo notar que las cantidades ϕ (r_j) son valores numéricos conocidos, puesto que son función de los valores de las entradas, centroides y factores de escala (cantidades todas ya conocidas), los pesos w_{kj} y umbrales θ_k se calculan simplemente aplicando el algoritmo de la Adalina *(Adaline)* mediante la regla LMS (*Least Mean Squares*, mínimos cuadrados promedio) [30] a la expresión de la salida de la capa final

$$z_k = \sum_j w_{kj}\phi(r_j) + \theta_k$$

que queda de la forma

$$w_{kj}(t+1) = w_{kj}(t) + \varepsilon\left(t_k - z_k\right)\phi(r_j)$$

siendo t_k los valores objetivo (*target*). Los umbrales se actualizan siguiendo el mismo esquema, considerando que se trata de pesos con entradas de valor -1.

Al utilizar el aprendizaje en dos etapas (cálculo de los parámetros de la capa oculta, cálculo de los de la capa de salida), se consigue acelerar notablemente el proceso de aprendizaje respecto del *BackPropagation***,** el cual opera globalmente (se suele indicar que el *BackPropagation* es unas tres veces más lento). Es muy importante tener en cuenta que de las dos etapas que conforman el aprendizaje, la primera hace uso de un algoritmo no supervisado, sea el de *Kohonen* o el de *k-medias*, mientras que en la segunda se emplea el algoritmo de la *Adalina*, de tipo supervisado. Al conjugar un tipo de aprendizaje no supervisado en la capa oculta, con otro supervisado en la de salida, en ocasiones se dice que este modelo es una red hibrida.

1.2.3 PSO (*Particle Swarm Optimization*).

La optimización mediante enjambre de partículas es un algoritmo evolutivo, introducido por primera vez en 1995 por *Kennedy* y *Eberhart* [36]. Se basa en el uso de una población (enjambre) de partículas para recorrer un espacio de soluciones de un problema en busca de la más eficiente. Éste término es caracterizado por una función de aptitud (*fitness*) específica del problema que se trate. La posición de la ***i***-ésima partícula en el espacio de búsqueda se representa por el vector (x_i), ésta se actualiza en cada instante de tiempo t a partir de su vector de velocidad (v_i). La velocidad se calcula en cada momento a partir de la velocidad anterior teniendo en cuenta la mejor posición visitada por la partícula (p_i) y la mejor posición global del enjambre (p_g). Las ecuaciones utilizadas en estos cálculos son:

$$v_{id}(t+1) = wv_{id}(t) + c_1 r_1 \left(p_{id} - x_{id}(t)\right) + c_2 r_2 \left(p_{gd} - x_{id}(t)\right)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1)$$

donde d es la dimensión de las partículas, w es el peso de inercia, c_1 y c_2 son constantes conocidas como coeficientes de aceleración que permiten establecer la importancia establecida para la mejor posición alcanzada por la partícula (p_{id}) y la mejor posición alcanzada por el enjambre (p_{gd}) respectivamente, r_1 y r_2 son dos números aleatorios uniformemente distribuidos en el rango de 0 a 1.

El algoritmo comienza inicializando de forma aleatoria la posición de las partículas dentro del espacio de búsqueda. Luego, en cada iteración, se calculan las nuevas posiciones de

las partículas sobre todas sus dimensiones a partir de la actualización del vector de velocidad, se evalúa la función de aptitud con cada nueva posición y se actualizan la mejor posición específica de cada partícula y la mejor global de todo el enjambre. El peso de inercia suele utilizarse variable, de forma tal que en las primeras iteraciones los pasos de búsqueda en el espacio sean grandes y permitan localizar rápidamente posibles mínimos y al final sea pequeño para que no existan oscilaciones en un posible final del mínimo.

El uso de PSO se ha extendido al entrenamiento de redes neuronales como se describe en la literatura [37-41]. En este caso, PSO considera la búsqueda del mejor conjunto de pesos, por tanto cada dimensión de las partículas se refiere a un vector de pesos de la red. La función de aptitud utilizada es el error cuadrático medio que genera el MLP sobre el conjunto de datos de entrenamiento.

Bajo el enfoque del entrenamiento de redes neuronales, a este algoritmo inicial o clásico, se le han efectuado numerosas modificaciones con el objetivo de resolver problemas como la lenta convergencia en determinados casos y la posibilidad de caer en mínimos locales de la función de aptitud.

En [42] se expone una modificación para la utilización de PSO en el entrenamiento de Redes Neuronales *Feedforward* denominada *Multi-Phase Particle Swarm Optimization* (MPPSO). Este algoritmo ayuda a realizar una búsqueda más amplia en el espacio de soluciones incrementando la diversidad de la población de partículas y previniendo convergencias prematuras. MPPSO utiliza múltiples grupos de partículas que cambian la dirección de la búsqueda en diferentes fases del algoritmo.

MPPSO difiere del PSO original en las ecuaciones utilizadas para la actualización de la posición y velocidad de cada partícula. Para la actualización del vector de velocidad se emplea:

$$v_{id}(t+1) = Cv * v_{id}(t) + Cg * p_{gd}(t) + Cx * x_{id}(t)$$

Este algoritmo solo permite cambios en la posición de la partícula que resulten en alguna mejora del valor de la función de aptitud; por lo tanto no se necesita el término de mejor posición de la partícula (p_{id}) como en el algoritmo clásico.

El proceso de actualización depende de los valores de los coeficientes ***Cv, Cg, y Cx***. Cada partícula se encuentra en un grupo y fase específico en un momento dado. Los coeficientes ***Cg,*** y ***Cx*** deben tener signos diferentes. Los signos indican si la partícula se mueve hacia o desde la mejor partícula global. El hecho de "perseguir" múltiples objetivos, utilizando diferentes partículas, incrementa la posibilidad de encontrar nuevas y mejores soluciones candidatas para el problema de optimización a ser resuelto. En algunos casos resulta más conveniente alejarse de la posible mejor solución que permanecer en su vecindad.

El número de fases y grupos, así como los objetivos de búsqueda para cada grupo y cada fase se deben seleccionar previamente a la ejecución del algoritmo.

Este algoritmo, como mecanismo para remediar la posibilidad de caer en mínimos locales de la función objetivo, utiliza la reinicialización periódica del vector de velocidad; lo implementa mediante un parámetro denominado *velocityChangeIteration* que establece la cantidad de iteraciones que ocurren entre cada reinicialización.

Otro interesante algoritmo propuesto a partir de la idea original ha sido descrito en [43] y nombrado como *Optimización Adaptativa basada en Enjambre de Partículas* o APSO (por sus siglas en inglés de *Adaptive Particle Swarm Optimization*). Fue concebido para utilizarse en espacios de búsqueda con una gran dimensionalidad y donde la función de error se caracteriza por grandes llanuras y pasos empinados como se muestra en la ***Figura 6***

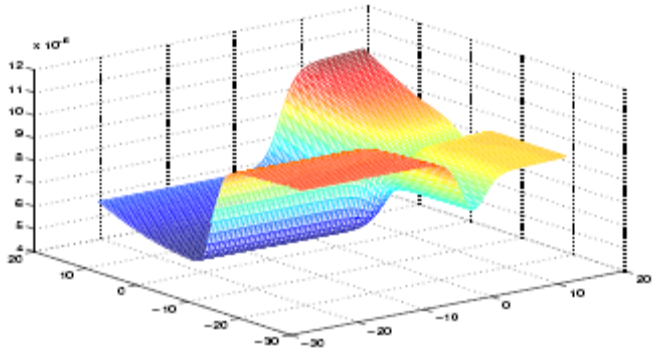


Figura 6*. Superficie de error caracterizada por grandes llanuras y pasos empinados.*

Este algoritmo introduce la concepción del *ritmo de aprendizaje variable* utilizado en el algoritmo de entrenamiento por *BackPropagation* de las redes neuronales. La idea surge a

partir de la siguiente observación. Una región empinada alrededor de un posible mínimo sugiere que las partículas estén "volando" directamente sobre estas posiciones. A pesar de ser un área de gran importancia puede que reciba, por parte del enjambre, una exploración relativamente pequeña. Esto puede ser posible por el hecho de que la población de partículas sea pequeña si la comparamos con el número de dimensiones del espacio de búsqueda. Además, como la función de error combina el valor de aptitud sobre todas las dimensiones, la partícula no es capaz de determinar en que dimensión se están logrando mejoras.

La idea de APSO es ajustar un término de *ritmo de aprendizaje* en el algoritmo de forma tal que en las regiones empinadas las nuevas posiciones que se calculen para las partículas, no "sobrevuelen" la posición que constituye el mínimo de la función de aptitud.

De esta forma la incorporación del parámetro *ritmo de aprendizaje* induce un cambio en las ecuaciones originales de PSO. Estas son consideradas como una versión discretizada de las leyes del movimiento de Newton con una unidad de paso de tiempo (*time step* ó ***dt***) y un término de aceleración (a_{id}) proporcional a las distancias entre la posición actual de la partícula y su mejor posición, así como entre dicha posición actual y la mejor posición de todo el enjambre. En esta concepción se considera además, el uso de un factor ***k*** como una constante negativa equivalente a un elemento de fricción; su función es la de mantener la cohesión del enjambre al limitar los valores que toman las dimensiones de las partículas. Las ecuaciones quedan:

$$a_{id}(t) = c_1 r_1 \left(p_{id} - x_{id}(t)\right) + c_2 r_2 \left(p_{gd} - x_{id}(t)\right) + k * v_{id}(t)$$

$$v_{id}(t + dt) = v_{id}(t) + a_{id} * dt$$

$$x_{id}(t + dt) = x_{id}(t) + v_{id}(t) * dt + \frac{a_{id}(t) * dt^2}{2}$$

El paso de tiempo ***dt***, se utiliza con la misma concepción que la del *ritmo de aprendizaje* en las redes neuronales, es decir, controla cuanto cambio ocurre en cada iteración. Así, el ajuste del parámetro ***dt,*** conduce a un ritmo de "aprendizaje" adaptativo en el algoritmo.

En APSO este ajuste se realiza según el siguiente algoritmo:

```
1: Inicializar la población
2: Determinar la mejor partícula de la población
3: Mientras no se cumpla el criterio de salida {
        Para cada partícula x {
                Calcular la aceleración en t
                Calcular la velocidad en t+dt
                Calcular la posición en t+dt
                Si (error(posición en t+dt) < error(posición en t)) {
                        posición = posición en t+dt
                        velocidad = velocidad en t+dt
                        dt(x) = dt(x) * stepInc
                } Sino {
                        Si (error(posición en t+dt) / error(posición) < maxErrInc) {
                                posición = posición en t+dt
                                velocidad = velocidad en t+dt
                        } Sino {
                                dt(x) = dt(x) * stepDec
                                velocidad = velocidad * stepDec
                        }
                }
        }
  }
```

Como se observa, a cada paso se calcula una nueva posición y su error. La nueva posición solo se acepta si su error es a lo máximo ***maxErrInc*** veces el error de la posición anterior. Si se rechaza la nueva posición, entonces el tamaño del paso (***dt***) se decrementa por el factor ***stepDec***, lo mismo se hace con la velocidad para prevenir que el valor de velocidad anterior domine en la contribución al nuevo valor de velocidad.

1.2.4 Máquinas de vectores de soporte para regresión (SVMR).

Las máquinas de vectores de soporte (SVM) fueron desarrolladas por *Vapnik* (1995) [44], para resolver problemas de clasificación. Su modificación, para ser utilizadas en

problemas de regresión (SVMR), fue desarrollada en los laboratorios de AT&T por *Vaptnik* [45]. SVMR está basado en el principio de minimización del riesgo estructural (SRM por sus siglas en inglés, *Structural Risk Minimization*), principio originado de la teoría de aprendizaje estadístico desarrollada por *Vapnik* en [44], el cual ha demostrado ser superior al principio de minimización del riesgo empírico (ERM por sus siglas en inglés, *Empirical Risk Minimization*), utilizado por las redes neuronales convencionales.

En los últimos tiempos este método ha ganado gran popularidad, siendo utilizado con éxito en varias aplicaciones [46-48]. Algunas de las razones de su creciente popularidad son:

- Excelente capacidad de generalización, debido a la minimización del riesgo estructural.
- Existen pocos parámetros a ajustar; el modelo solo depende de los datos con mayor información.
- La estimación de los parámetros se realiza a través de la optimización de una función de costo convexa, lo cual evita la existencia de un mínimo local.
- La solución es *sparse*, lo cual significa que la mayoría de las variables son cero en la solución, por lo que el modelo final puede ser escrito como una combinación de un número muy pequeño de vectores de entrada, llamados vectores de soporte.

Una descripción detallada del método SVMR puede ser revisada en [49]. A continuación se describirán algunas consideraciones que serán útiles a los efectos de analizar los resultados obtenidos.

En SVMR la idea básica consiste en realizar un mapeo de los datos de entrenamiento $x \in X$, a un espacio de características de mayor dimensión F a través de un mapeo no lineal $\phi : X \rightarrow F$, donde se puede realizar una regresión lineal.

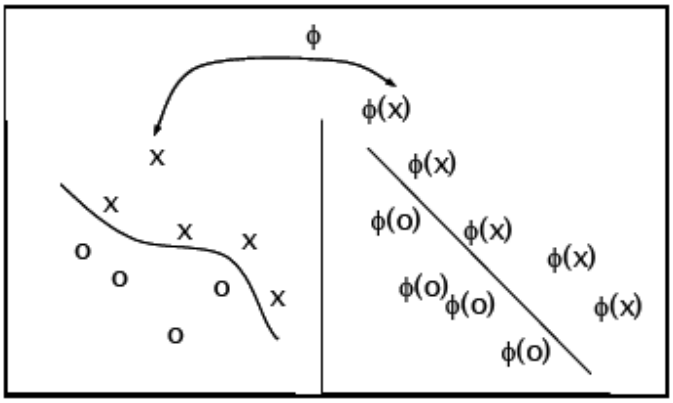


***Figura 7**. Mapeo del espacio de entradas a un espacio de características de mayor dimensión.*

El empleo de una función *kernel* permite realizar este mapeo de forma implícita mediante el producto punto sin tener que calcular explícitamente el mapeo $\phi : \mathrm{X} \rightarrow \mathrm{F}$. Se define un kernel K como una función, tal que para todo $x, z \in \mathrm{X}$

$$K(x,z) = \langle \phi(x) \cdot \phi(z) \rangle = \sum_{i=1}^{l} \phi_i(x)\phi_i(z),$$

donde $\langle \cdot \rangle$ es el producto interno o producto punto

Algunas de las funciones *kernel* que más se utilizan son:

- *Polinomial*. Donde la función *kernel* toma la forma

$$k(x,z) = \left(\langle x \cdot z \rangle + c\right)^p \text{ con p}\in N, \text{ c > 0}$$

- *Funciones de Base Radial (**R**adial **B**asis **F**unction).* La función *kernel* toma la forma

$$k(x,z) = \exp(-\| x - z \|^2 / \sigma^2)$$

En el ***Anexo 1*** se refieren los kernels implementados en WEKA hasta su versión 3.5.5

En SVMR la meta es encontrar una función *f(x)* que tenga a lo más una desviación ***ε*** de la salida y_i para todos los datos de entrenamiento, y al mismo tiempo, que sea lo más mínima posible. Es decir, se utiliza una función de costo o pérdida como la siguiente:

$$c\left(x, y, f\left(x\right)\right) = \left|y - f\left(x\right)\right|_{\varepsilon} = \max\left\{0, \left|y - f\left(x\right)\right| - \varepsilon\right\}$$

que también se conoce como Vapnik's ε *-insensitive loss function*

Se parte de pretender aproximar un conjunto de entrenamiento $\{x_i, y_i\}$, $x \in R^n$, $y \in R$ por medio de una función lineal $f : X \supseteq R^n \rightarrow R$,

$$f(x) = \langle w \cdot x \rangle + b$$

$$= \sum_{i=1}^{n} w_i x_i + b$$

donde, $x = \{x_1, x_2, \cdots, x_n\}$, $y = \{y_1, y_2, \cdots, y_n\}$ y $w = \{w_1, w_2, \cdots, w_n\}$ siendo w el vector de pesos y b es el bias. Se busca encontrar w lo más mínima posible y para ello, una forma es minimizar la norma Euclidiana $||w||^2$.

De tal forma se trata de minimizar

$$\frac{1}{2} \| w \|^2 + C \sum_{i=1}^{m} |y_i - f(x)|_\varepsilon$$

donde la constante $C > 0$ determina el compromiso entre la llanura de la función f y la cantidad hasta la cual pueden ser toleradas desviaciones mayores que ε. Es decir, se asume que existe la función $f(x)$ que es capaz de aproximar a todos los pares (x_i, y_i) con una precisión ε. Algunas veces este no es el caso por lo que se introducen las variables de holgura para cuando $f(x_i) - y_i > \varepsilon$ y cuando $y_i - f(x_i) > \varepsilon$, que se denotan como ξ y ξ^*, y en conjunto como $\xi^{(*)}$

Por tanto queda el siguiente problema de optimización:

$$\text{minimizar } \tau\left(w, \xi^{(*)}\right) = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{m} \left(\xi_i + \xi^*\right) \tag{5}$$

sujeto a: $f(x_i) - y_i \leq \varepsilon + \xi_i$

$$y_i - f(x_i) \leq \varepsilon + \xi_i^*$$

$$\xi_i \, , \, \xi_i^* \geq 0$$

Lo anterior se convierte en un problema de programación cuadrática que se puede resolver introduciendo multiplicadores de *Lagrange*. El problema generalizado mediante el empleo de *kernels* quedaría entonces como:

Dados $C, \varepsilon \geq 0$

maximizar $W(\alpha, \alpha^*) = -\varepsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) y_i - \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(x_i, x_j)$

sujeto a $0 \leq \alpha_i, \alpha_i^* \leq C$ para toda *i* = 1, …, *m* y $\sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0$

La solución de esta problemática se ha implementado mediante varios algoritmos. Entre los más eficientes se encuentra el basado en el método de optimización mínima secuencial (SMO por sus siglas en inglés de *Sequential Minimal Optimization*) que plantea dividir el problema cuadrático en una serie de problemas cuadráticos de menor dimensión los cuales pueden ser resueltos de manera analítica evitando así utilizar técnicas numéricas. Este método fue introducido en [50] y se le han hecho mejoras como las referidas en [51], este último trabajo precisamente constituye la base del algoritmo implementado en WEKA y utilizado en el presente trabajo.

1.2.5 Procesos Gaussianos.

Desde la publicación de los primeros trabajos sobre el aprendizaje supervisado en redes neuronales, surgió un gran interés en la modelación empírica de relaciones entre datos de gran dimensionalidad utilizando modelos paramétricos no lineales como es el caso del Multilayer Perceptron y las funciones de base radial. En la interpretación Bayesiana de estos métodos se asume la existencia de una función no linear *y(x)* parametrizada según $\boldsymbol{w}$ que se ajusta y es inferida a partir de N datos $\{\mathbf{x}^{(n)}, \mathbf{t_n}\}$ (n =1, .., N). Dicha inferencia $y(x)$ se describe según la distribución de probabilidad:

$$P(y(x)|t_N, X_N)) = \frac{P(t_N|y(x), X_N) P(y(x))}{P(t_N|X_N)}$$

De los términos de la derecha, el primero, $P(t_N|y(x), X_N)$, es la probabilidad de los valores observados (t_N) dada la función $y(x)$, la cual en problemas de regresión se asume regularmente como una distribución Gaussiana separable; el segundo término, $P(y(x))$ es la distribución de probabilidad previa (conocida como *prior*) sobre las funciones que asume

el modelo. Dicha *prior* está implícita en la selección del modelo paramétrico y en la selección de los regularizadores utilizados en el proceso de adaptación del modelo.

Desde el punto de vista del proceso de predicción de futuros valores de $\boldsymbol{t}$, todo lo que importa es la asunción de $P(y(x))$ y el nivel de ruido del modelo. La parametrización de la función $y(x; w)$ es irrelevante.

La idea de la modelación de los Procesos Gaussianos es establecer $P(y(x))$ directamente en el espacio de funciones sin tener que parametrizar $y(x)$. La más simple de las distribuciones previas sobre las funciones se le denomina Proceso Gaussiano. Se puede considerar como la generalización de una distribución Gaussiana sobre un espacio vectorial finito a un espacio de función de dimensión infinita. Como una distribución Gaussiana se especifica a partir de su media y su matriz de covarianza, un Proceso Gaussiano también se especifica mediante una media y una función de covarianza. Aquí la media es una función de $\boldsymbol{x}$ (frecuentemente se utiliza la función cero), y la covarianza es una función $C(x, x')$ la cual expresa la covarianza esperada entre el valor de la función $\boldsymbol{y}$ en los puntos $\boldsymbol{x}$ y $\boldsymbol{x'}$.

Regresiones no lineales

Dados $\boldsymbol{N}$ puntos de datos X_N, $t_N = \{x^{(n)}, t_n\}^N_{n=1}$ donde las entradas $\boldsymbol{x}$ son vectores de dimensión $\boldsymbol{I}$ y los valores $\boldsymbol{t}$ de la clase son números reales; si asumimos que la función $y(x)$ se ajusta a los datos observados, la tarea es inferir dicha función a partir de los datos y predecir su valor (o el valor de la observación t_{N+1}) en nuevos puntos $\boldsymbol{x}^{(N+1)}$.

Considerando un acercamiento paramétrico a la problemática, se parte de que la función desconocida $y(x)$ se expresa en términos de una función no linear $y(x; w)$ parametrizada según $\boldsymbol{w}$.

Ejemplo 1: Funciones bases fijas. Utilizando un conjunto de funciones bases $\{\phi_h(x)\}_{h=1}^H$, se puede expresar

$$y(x;w) = \sum_{h=1}^{H} w_h \phi_h(x)$$

Si las funciones bases son funciones no lineales de x como es el caso de las funciones de base radial centradas en puntos fijos $\{c_h\}_{h=1}^{H}$,

$$\phi_h(x) = \exp\left[-\frac{(x-c_h)^2}{2r^2}\right]$$

entonces *y(x; w)* será una función no linear de ***x***.

Otros conjuntos de funciones bases fijas pueden incluir, por ejemplo, polinomios de la forma $\phi_h(x) = x_i^p x_j^q$ donde *p* y *q* son potencias enteras que dependen de *h*.

Ejemplo 2: Funciones de base adaptativa. Alternativamente, se puede establecer la función *y*(x) a partir de funciones bases que dependan de parámetros adicionales incluidos en el vector ***w***. En una red neuronal *feedforward* de tres capas con nodos ocultos nolineales y un nodo lineal de salida, la función puede definirse como:

$$y(x;w) = \sum_{h=1}^{H} w_h^{(2)} \tanh\left(\sum_{i=1}^{I} w_{hi}^{(1)} x_i + w_{h0}^{(1)}\right) + w_0^{(2)}$$

Donde ***I*** es la dimensionalidad del espacio de entrada y el vector de pesos ***w*** está compuesto por pesos de entrada $\{w_{hi}^{(1)}\}$, los bias de los nodos ocultos $\{w_{h0}^{(1)}\}$, los pesos de salida $\{w_h^{(2)}\}$ y el bias del nodo de salida $w_0^{(2)}$.

La función *y(x; w)* se obtiene entonces a partir de la inferencia de ***w***. La mayoría de los métodos para inferir parámetros pueden ser interpretados en términos de un modelo Bayesiano para el problema, en el cual la distribución posterior de los parámetros está dada por:

$$P(w|t_N, X_N)) = \frac{P(t_N|w, X_N)P(w)}{P(t_N|X_N)}$$

El factor $P(t_N|X_N)$ establece la probabilidad de los puntos de datos observados cuando los parámetros ***w*** (y por tanto la función ***y***) son conocidos. Esta distribución de probabilidad se toma como una distribución Gaussiana separable, cada punto de dato t_n difiere del valor estimado $y(x^{(n)}; w)$ a partir de un ruido adicional. El factor $P(w)$ especifica la distribución de probabilidad previa de los parámetros. Ésta también se toma como una distribución

Gaussiana separable. Si la dependencia de ***y*** sobre ***w*** es no linear, la distribución posterior $P(w|t_N, X_N))$ no es, en general, una distribución Gaussiana.

La inferencia se pude implementar de varias formas. Una técnica es minimizar la función objetivo

$$M(w) = -log[P(t_N|w, X_N)P(w)]$$

con respecto a ***w***, determinando localmente los parámetros más probables y utilizando entonces la curvatura de ***M***, $\partial^2 M(w)/\partial_{wi}\partial_{wj}$ para definir barras de error sobre ***w***. Un método más general es utilizar la técnica de *Markov-Monte Carlo* para crear ejemplos desde la distribución posterior $P(w|t_N, X_N)$.

Habiendo obtenido una de estas representaciones de la inferencia de ***w*** a partir de los datos, las predicciones se efectúan por marginalización sobre los parámetros:

$$P(t_{N+1}|t_N, X_{N+1}) = \int d^H w\, P\left(t_{N+1}\middle| w, x^{(N+1)}\right) P\left(w|t_N, X_N\right)$$

Si se ha encontrado una representación Gaussiana de la distribución posterior $P(w|t_N,X_N)$, entonces la integral anterior típicamente puede ser evaluada directamente. En la alternativa de Monte Carlo, la cual genera ***R*** ejemplos $\mathbf{w}^{(r)}$ considerados provenientes de la distribución posterior $P(w|t_N, X_N)$, la distribución predictiva se aproxima según

$$P(t_{N+1}|t_N, X_{N+1}) = \approx \frac{1}{R}\sum_{r=1} P\left(t_{N+1}\middle| w^{(r)}, x^{(N+1)}\right)$$

En [52] se aborda además la problemática desde un acercamiento no paramétrico de la problemática.

En general un Proceso Gaussiano se puede definir como una distribución de probabilidad sobre un espacio de funciones *y*(x) que puede ser escrita como:

$$P\left(y(x)\middle|\mu(x), A\right) = \frac{1}{Z}\exp\left[-\frac{1}{2}\left(y(x)-\mu(x)\right)^T A\left(y(x)-\mu(x)\right)\right],$$

donde $\mu(x)$ es la función de la media de la distribución y *A* es un operador linear y donde el producto punto de dos funciones $y(x)^T z(x)$ se define, por ejemplo, según $\int dx.y(x)z(x)$.

Igualmente se puede definir que una distribución de probabilidad de una función y(x) es un proceso Gaussiano si para cualquier selección finita de puntos $x^{(1)}, x^{(2)},\ldots, x^{(N)}$, la densidad marginal $P(y(x^{(1)}), y(x^{(2)}), \ldots, y(x^{(N)}))$ es una distribución Gaussiana.

Una amplia referencia sobre este modelo y la comparación con otras técnicas puede ser revisada en [53].

1.3 Conclusiones Parciales.

A partir del estudio de la bibliografía consultada se puede observar que la estimación de la tensión de ruptura por creep mediante métodos empíricos ha constituido, en el transcurso de la historia del desarrollo de nuevas aleaciones, un elemento de amplio estudio. Modelos recientes, desarrollados dentro del campo del aprendizaje estadístico y el aprendizaje automatizado como SVMR y Procesos Gaussianos, no han sido empleados en esta problemática, a pesar de que por su concepción teórica apuntan a resultados eficientes.

CAPÍTULO 2.

Experimentación con modelos de estimación de funciones continuas para el pronóstico de la tensión de ruptura por creep.

En el presente trabajo, a partir de la experiencia en el pronóstico de la tensión de ruptura por *creep* mediante métodos empíricos donde se destacan los resultados obtenidos por el modelo basado en redes neuronales propuesto por *T.Sourmail*, se diseñaron experimentos que permiten comparar el comportamiento de varios modelos de estimación de funciones continuas en la problemática del *creep*. Se utilizaron métodos, desde aquellos tan convencionales como las regresiones lineales hasta otros de más reciente desarrollo y utilización, como es el caso de las Redes Neuronales, las Máquinas de Vectores de Soporte para Regresión, los Procesos Gaussianos, entre otros.

2.1 Características de los datos utilizados en los experimentos.

Como es conocido, un método empírico obtiene un modelo ajustando sus parámetros (entrenamiento) a partir de una base de casos o ejemplos de los cuales debe "aprender". De aquí resulta obvia la importancia de los datos. Estos deben ser lo más representativos posible dentro del dominio del problema que se desea modelar.

En el presente trabajo se decidió realizar el estudio del comportamiento de la tensión de ruptura por *creep*, a partir de las mismas variables utilizadas en los modelos referenciados en los trabajos [27-29] y que se obtuvieron con redes neuronales; esto motivado principalmente por

- los resultados satisfactorios obtenidos con dichos modelos y
- permitir establecer una comparación de resultados respecto a estos trabajos.

La cantidad de variables a utilizar puede ser motivo de varios análisis; sí es importante señalar que:

- un aumento del número de variables, probablemente conduzca a una reducción de los datos que se obtienen de la literatura publicada;
- los datos sobre la composición química, los tratamientos térmicos y el tiempo de servicio constituyen la mínima información necesaria para estimar el *creep* pues la microestructura de los aceros depende críticamente de estas variables y

- a los efectos del presente trabajo estas variables satisfacen las necesidades pues los experimentos serán desarrollados consistentemente (con los mismos datos y las mismas variables) para todos los métodos.

Para los experimentos se utilizó una recopilación de casos que se encuentran expuestos en el sitio web MAP[1]. Estos datos, según MAP, fueron compilados por el Dr Takahashi Yoshida (de *Ishikawajima Harima Heavy Industries*, Japón) a partir de literatura publicada y referenciada en [28]. Consiste de 2066 casos que muestran los valores de la tensión de ruptura por *creep* relacionados con 37 variables y compuestos principalmente por valores de dos tipos de aceros típicos: el *Fe–2.25Cr–1Mo* y el *Fe-(9-12)Cr*. Se considera una fuente de datos muy representativa del dominio del problema que se trata.

La ***Tabla 1*** muestra las variables utilizadas y algunas de sus características.

Variable	Rango	Media	Desv. Stand.
Condiciones de trabajo			
Log (tiempo, h)	−0·22 to 5·28	3·021	1·009
Temperatura, K	723–977	866·6	61·6
Composición química			
C,wt-%	0·0040–0·230	0·1120	0·0440
Si,wt-%	0·0100–0·860	0·2900	0·1700
Mn,wt-%	0·2700–0·920	0·5100	0·1100
P,wt-%	0·0010–0·029	0·0130	0·0076
S,wt-%	0·0010–0·020	0·0076	0·0047
Cr,wt-%	2·1700–12·90	8·4300	3·2800
Mo,wt-%	0·0400–2·990	0·8900	0·5130
W,wt-%	0·0100–3·930	0·4100	0·7490
Ni,wt-%	0·0100–2·000	0·2400	0·2800
Cu,wt-%	0·0100–0·870	0·0740	0·1020
V,wt-%	0·0100–0·280	0·1190	0·1000
Nb,wt-%	0·0050–0·312	0·0360	0·0470
N,wt-%	0·0010–0·165	0·0310	0·0273
Al,wt-%	0·0010–0·057	0·0120	0·0120
B,wt-%	0·0003–0·051	0·0010	0·0040
Co,wt-%	0·0080–2·500	0·0920	0·3340
Ta,wt-%	0·0003–0·100	0·0008	0·0069
O,wt-%	0·0030–0·035	0·0104	0·0026
Re,wt-%	0·0003–0·600	0·0032	0·0416

Variable	Rango	Media	Desv. Stand.
Tratamientos Térmicos			
Normalising			
Temperatura, K	1123–1453	1279	70·42
Duración, h	0·17–33	2·007	3·85
Enfr en Horno	0 – 1	0.060	0.24
Aire	0 – 1	0.580	0.49
Aceite	0 – 1	0.250	0.43
Agua	0 – 1	0.110	0.31
Tempering			
Temperatura, K	823–1133	980	71·8
Duración, h	0·5–32	3·34	5·88
Enfr en Horno	0 – 1	0.060	0.24
Aire	0 – 1	0.880	0.33
Aceite	0 – 1	0.030	0.17
Agua	0 – 1	0.32	0.18
Annealing			
Temperatura, K	0–1023	230·7	404·3
Duración, h	0·5–50	3·96	8·15
Enfr en Horno	0 – 1	0.054	0.226
Aire	0 – 1	0.946	0.226

Tabla 1. *Variables de entrada utilizadas. Características.*

[1] Siglas en inglés de *Materials Algorithms Project*, proyecto desarrollado por el Laboratorio Nacional de Física y la Universidad de Cambridge, Reino Unido. http://www.msm.ac.uk./map/map.html

En el ***Gráfico 1*** se muestra una distribución de la cantidad de ejemplos en base a rangos de valores de la variable que se desea estimar (la *tensión de ruptura por creep*)

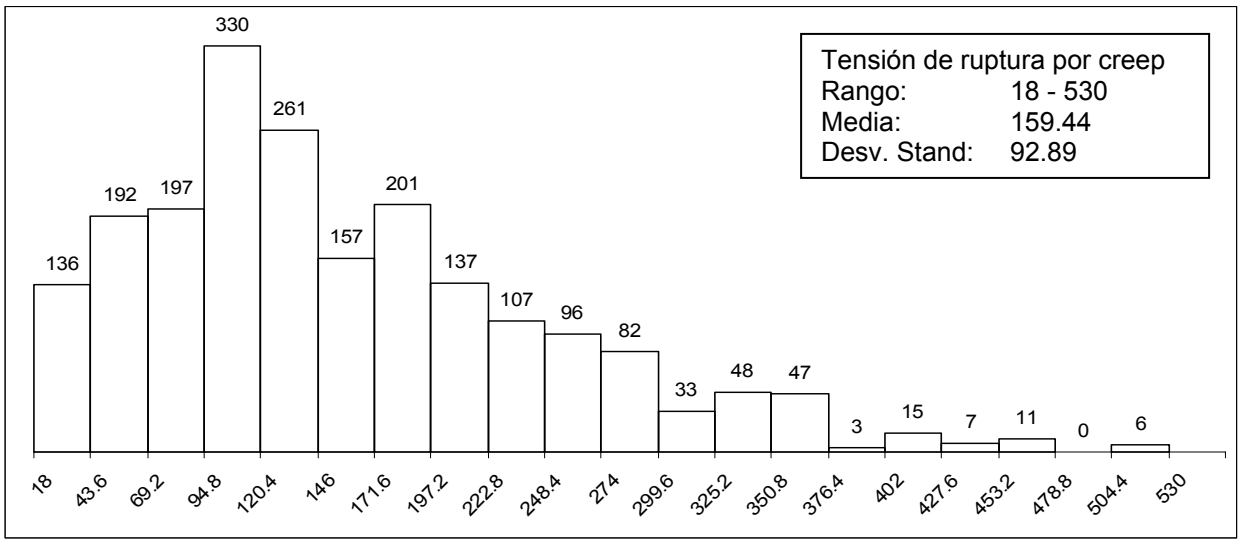


Gráfico 1. *Distribución de ejemplos en base a los valores de tensión de ruptura por creep (el eje X representa los valores de tensión y el eje Y la cantidad de ejemplos con valores comprendidos en el rango representado por el eje X)*

2.2 Diseño de los experimentos.

El primer aspecto que resulta de esencial importancia para garantizar resultados sólidos, es la organización de los datos. En el presente trabajo se utilizó siempre un 80 por ciento de los casos para obtener el modelo mediante el proceso de entrenamiento y el 20 por ciento restante para comprobar como estos modelos estiman la tensión de ruptura por creep en datos no "vistos" anteriormente. A los efectos de terminología, se denominará a estos conjuntos de datos como de *entrenamiento* y *prueba* respectivamente.

La división del conjunto original de ejemplos en 1653 casos (el 80 %) para entrenamiento y 413 casos (el 20%) para prueba se realizó de forma aleatoria. Este proceso de división aleatoria se efectuó 30 veces obteniéndose las 30 bases de casos utilizadas en los experimentos.

2.2.1 Modelos empleados.

Para efectuar el estudio de los datos de creep se utilizaron nueve métodos de estimación de funciones, experimentándose en la mayoría de los casos, con diferentes configuraciones. Los métodos y configuraciones se relacionan en la ***Tabla 2***.

Métodos	Configuraciones
1. Procesos Gaussianos. * Los detalles de este método pueden ser revisados en [47].	Se utilizaron los siguientes *Kernels*: a. *Kernel* basado en RBF b. *Kernel* Polinomial (grado 2) c. *Kernel* Polinomial Normalizado (grado 2)
2. Regresión Isotónica. *	
3. Regresión lineal por mínimos cuadrados medio. (Least Median Square regression) *	
4. Regresión lineal. *	Se utilizaron varios métodos de selección de atributos: a. sin selección de atributos b. M5 c. Selección Golosa utilizando la métrica de información de Akaike
5. *Pace Regression*. *	Se utilizaron los siguientes estimadores: a. Empirical Bayes b. Optimal subset selector for normal mixture. c. PACE2 for Chi-square mixture. d. PACE6 for Chi-square mixture. e. AIC estimator
6. PLS (Partial Least Square) Classifier. *	
7. RBF (Radial Basis Function) Network. *	Se experimentó con diferentes cantidades de cluster
8. Maquina de vectores de soporte para regresión. (SVMR). *	Se utilizaron los siguientes kernels (C = 1.0) a. *Kernel* Polinomial Normalizado (grado 2) b. *Kernel* Polinomial (grado 2) c. *Kernel* basado en RBF
9. MLP con Backpropagation*	Se experimentó con varias cantidades de nodos en la capa oculta. Desde 10 hasta 40, con 50 y con 60.
10. MLP con PSO	Se corrió para 11 nodos en la capa oculta.

Tabla 2. Métodos y configuraciones empleadas en los experimentos.
(*) Para obtener los resultados se utilizó la implementación de WEKA 3.5.5 de estos métodos.

La aplicación de estos métodos se logró a partir de la implementación que tiene de los mismos, el ambiente de trabajo WEKA en su versión 3.5.5. En la ayuda de este sistema se puede encontrar una breve descripción de los mismos.

2.2.2 Estimador del error.

Para cuantificar el comportamiento de los métodos se utilizó una medida del error en la estimación de la tensión de ruptura por *creep* en el conjunto de datos de prueba comparando los valores estimados de salida $\boldsymbol{y_\mu}$ con los valores provenientes de las mediciones experimentales $\boldsymbol{t_\mu}$. Específicamente se empleó la raíz del error cuadrático medio (***rmse*** – *root mean squared error*) que brinda valores en el orden de los datos que se desean estimar. Este indicador se obtiene como:

$$rmse = \sqrt{\frac{\sum_{\mu=1}^{m} \left(t_\mu - y_\mu\right)^2}{m}},$$

donde *m* es la cantidad de ejemplos del conjunto de prueba.

2.2.3 Procesamiento estadístico.

Se utilizó el test de Friedman para obtener un ranking de los modelos y luego el test de Wilcoxon para realizar una comparación par a par de los resultados obtenidos y determinar si existían diferencias significativas. Para el cálculo más exacto de todas las significaciones se utilizó la técnica de Monte Carlo que simula aleatoriamente 10000 muestras con distribución similar a la de los datos que se comparan y permite entonces estimar la significación promedio y un intervalo de confianza de la misma del 99%. Se consideró como significativo un valor promedio de la significación menor de 0.05.

2.2.4 Caracterización de los experimentos.

Los experimentos se dividieron en dos momentos.

El primer momento se dedicó a la obtención de la mejor configuración dentro de cada modelo. Para cada posible configuración se efectuaron 30 corridas del modelo correspondiendo con cada una de las 30 bases de casos disponibles. El valor de *rmse* de cada corrida fue promediado obteniendo así el indicador a comparar entre las configuraciones de un mismo modelo. Teniendo en cuenta el procesamiento estadístico se determinó, para cada modelo, la mejor configuración.

La segunda etapa del experimento tuvo como objetivo determinar el modelo de mejor comportamiento. Para ello se refinaron las configuraciones “vencedoras” en la primera etapa (se hizo una selección de los mejores modelos y dentro de ellos, su mejor configuración en base a los valores de *rmse*) y se efectuaron nuevamente 30 corridas de cada una. Al igual que en el primer momento del experimento, se obtuvo el promedio del valor de *rmse* de las corridas de cada modelo y se procesó estadísticamente para determinar el modelo de mejor comportamiento.

Las características de los experimentos, así como los parámetros utilizados en los diferentes modelos se detallan en el resto del epígrafe.

Optimización mediante enjambre de partículas (PSO)

En la presente investigación, la utilización de PSO se enfocó en tres objetivos:

1. Entrenamiento de un MLP.
2. Optimización de parámetros de un MLP.
3. Regresión lineal.

Se implementó una plataforma de clases que de forma general, permite abordar los objetivos referidos. Las características fundamentales del diseño implementado e integrado a WEKA se muestran en el siguiente diagrama de clases:

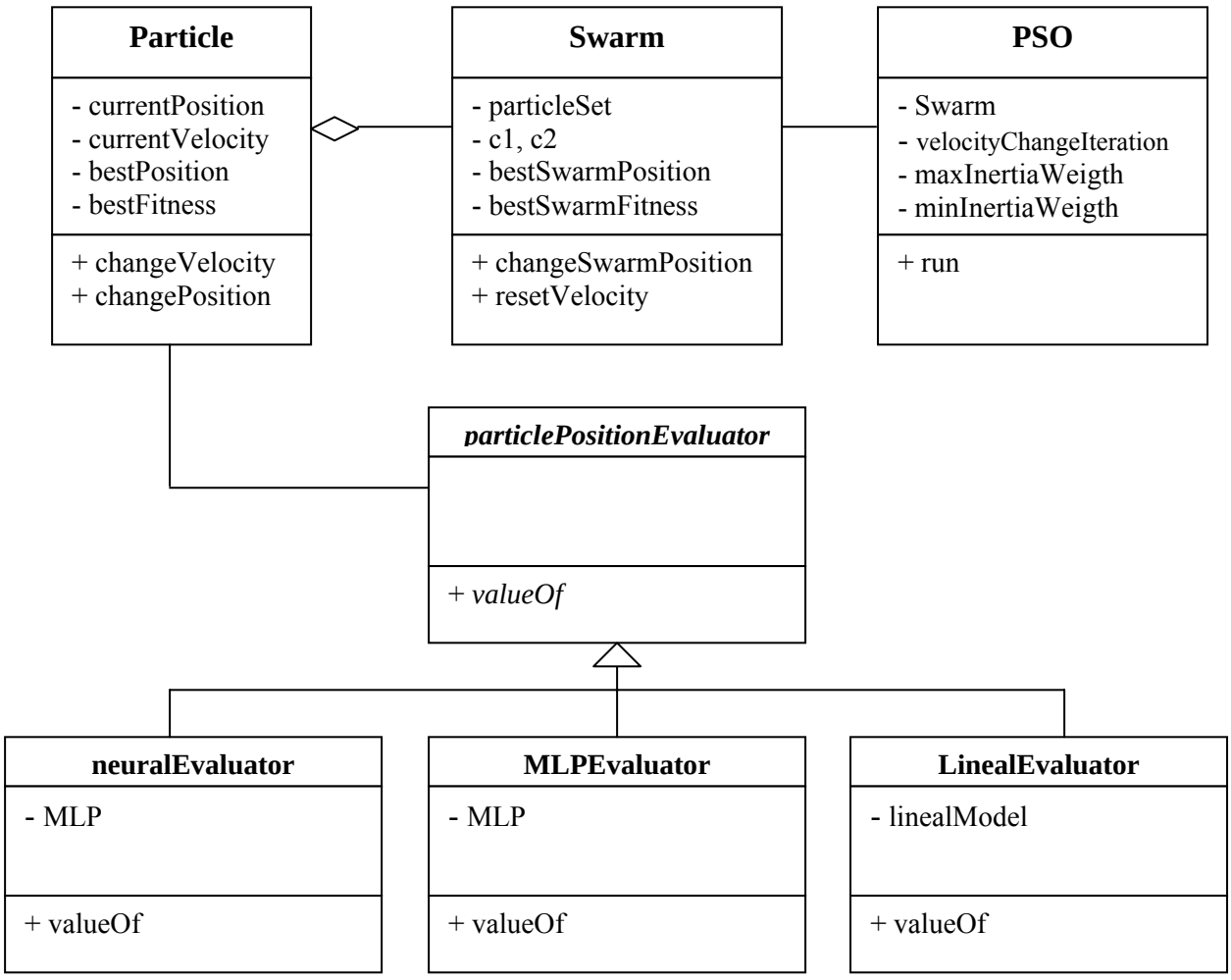


***Diagrama 1**. Diagrama de clases de la implementación de PSO en WEKA*

Esta plataforma de clases permite reutilizar el código implementado para cualquier problema donde se necesite realizar un proceso de optimización utilizando *PSO*. Solo es necesario implementar una nueva clase que herede de la clase abstracta (***ParticlePositionEvaluator***) e implementar el método ***valueOf*** que evalúa la posición de una partícula, es decir, evalúa la función de *fitness* elegida en el proceso de optimización a realizar por PSO para el problema específico.

La clase ***Particle*** posee información sobre el vector de posición actual de la partícula (*currentPosition*), el vector de velocidad actual (*currentVelocity*), el vector que representa la mejor posición de la partícula (*bestPosition*) al cual se asocia el mejor valor de *fitness* (*bestfitness*). Además implementa los métodos necesarios para:

- el cálculo del nuevo vector de velocidad (método *changeVelocity*) y
- el cálculo del nuevo vector de posición (método *changePosition*).

La clase ***Swarm*** posee información general para todo el enjambre: coeficiente de importancia de la mejor posición dentro de la partícula (*c1*), coeficiente de importancia de la mejor posición del enjambre (*c2*), mejor posición lograda por el enjambre (*bestSwarmPosition*), mejor valor de *fitness* logrado por el enjambre (*bestSwarmFitness*) y contiene a todo el conjunto de partículas (*particleSet*). Entre las responsabilidades principales que implementa están:

- cambiar la posición del enjambre (*changeSwarmPosition*) lo cual implica cambiar la posición de cada partícula,
- resetear el vector de velocidad (*resetVelocity*) de cada partícula; esta función se implementa para reducir la posibilidad de mínimos locales,

La clase ***PSO*** es quien controla la ejecución del proceso de optimización (método *run*) y cuando resetear el vector velocidad a partir del parámetro general *velocityChangeIteration*. Además posee información sobre los valores máximo y mínimo del peso de inercia a utilizar (*maxInertiaWeigth*, *minInertiaWeigth*).

En la investigación se crearon tres clases del tipo *ParticlePositionEvaluator* para resolver los tres enfoques en los que se utilizó PSO.

- La clase ***NeuralEvaluator*** se creó para utilizar PSO en el entrenamiento del MLP. En este caso la posición de una partícula se refiere a un conjunto de pesos de la red, el cual constituye el objeto de optimización. La función de *fitness* implementada en el método ***valueOf*** se corresponde con el error cuadrático medio que resulta de evaluar la red con el conjunto de pesos referidos por una partícula en específico.
- La clase ***MLPEvaluator*** permite utilizar PSO para optimizar el conjunto de parámetros de un MLP (en el caso de los experimentos se consideró un *MLP* con una sola capa oculta y como parámetros a optimizar: el momento y la cantidad de nodos ocultos, aunque el diseño permite incluir otros parámetros como cantidad de capas ocultas, etc). La posición de cada partícula se refiere a un conjunto de valores de estos parámetros. En este caso, la función de *fitness* obtiene el error cuadrático medio de la evaluación de los casos o ejemplos de prueba en el MLP cuyos parámetros son representados por la posición de la partícula que se evalúa en ese instante. Esta implementación permitió crear un nuevo clasificador en WEKA denominado ***PSO-MLP*** que utiliza PSO para obtener un modelo de red neuronal optimizado y emplea la implementación del modelo de MLP de WEKA para el entrenamiento y diagnóstico del MLP.
- La clase ***LinealEvaluator*** permite utilizar PSO para obtener una regresión lineal. Se parte de un modelo paramétrico de la forma $\boldsymbol{y = wx + b}$ siendo $\boldsymbol{w}$ el vector de pesos, $\boldsymbol{x}$ el vector de entradas y $\boldsymbol{b}$ el bias que permite realizar traslaciones al modelo lineal. PSO se utiliza, al igual que en el primer caso, para buscar el vector de pesos óptimo por lo que el vector de posición de una partícula representa un conjunto de pesos en específico. La función de *fitness* implementada se corresponde con el cálculo del error cuadrático medio que resulta de evaluar el modelo lineal caracterizado por un vector de pesos determinado, en el conjunto de entrenamiento. Este diseño permite contar con un nuevo clasificador denominado ***PSO-LinealRegression.***

Para la implementación de PSO en los tres enfoques, se utilizó el algoritmo clásico de este método pero utilizando la modificación propuesta en MPPSO de resetear el vector de velocidad cada cierto número de iteraciones con el objetivo de evitar mínimos locales. En lo adelante llamaremos a esta versión como PSO1.

Para el caso del entrenamiento del MLP, se implementó además el algoritmo APSO con el fin de lograr mejor rendimiento en cuanto a valores de error.

En la optimización de los parámetros de un MLP la función de *fitness* debe entrenar una configuración representada por la posición de una partícula y luego obtener el error cuadrático medio que genera dicha red entrenada sobre el conjunto de datos de prueba. Como se planteó anteriormente la posición de la partícula se refiere al momento y la cantidad de nodos ocultos; en este caso no se consideró el ritmo de aprendizaje pues el propio algoritmo implementado en WEKA establece un mecanismo de adaptación de este parámetro. Respecto a la cantidad de nodos ocultos se consideró una cantidad máxima de 60 a partir del conocimiento previo que se tiene sobre el comportamiento del parámetro en estos datos de *creep*.

Los valores de los parámetros utilizados en las diferentes implementaciones de PSO se relacionan a continuación:

De forma general, tanto para PSO1 como para APSO se tomaron los siguientes valores

Parámetro	Valor
Cantidad de partículas	• entrenamiento *MLP*: 30 • *PSO-MLP*: 20 • *PSO*-LinealRegression: 30
Cantidad de iteraciones	• entrenamiento *MLP*: 20000 • *PSO-MLP*: 50 • *PSO*-LinealRegression: 15000
Coeficiente de aceleración *c1*	2
Coeficiente de aceleración *c2*	2
velocityChangeIteration	10

En PSO1 además se utilizó

maxInertiaWeigth	0.5
minInertiaWeigth	0.2

En APSO además se utilizó

k	-2
dt inicial	5
stepInc	4.0
stepDec	0.05
MaxErrInc	1.3

Multilayer Perceptron (MLP).

En la primera etapa de los experimentos se determinó la mejor configuración del modelo a utilizar. Se empleó la implementación del clasificador *PSO-MLP* diseñado en la presente investigación con los valores de parámetros mencionados anteriormente. En este caso se entrenaron 1000 redes utilizando como parámetros de WEKA los siguientes:

Parámetro	Valor
Cantidad de Nodos Oculto	entre 1 y 60
Momento	entre 0 y 1
Ritmo de Aprendizaje	0.9 (con decadencia)
Iteraciones para entrenar una red	250

En la segunda etapa del experimento se utilizó un MLP con 11 nodos en la capa oculta. En este caso se utilizaron 10000 iteraciones en el entrenamiento con BackPropagation y 20000 con PSO.

Redes de funciones de base radial.

La implementación de WEKA utiliza el algoritmo *k-means* para la obtención de los centroides. Se experimentó con los siguientes valores

Parámetro	Valor
Desviación Standard Mínima para los clusters	3.6
Cantidad de Cluster para *k-means*	Múltiples valores entre 2 y 250

Máquinas de vectores de soporte para regresión (SVMR)

Como se planteó en el Capítulo 1, el algoritmo utilizado para la experimentación con este método es el SMO (*Sequential Minimal Optimization*) implementado en WEKA. Los valores de los parámetros utilizados en el primer momento de los experimentos fueron:

Parámetro	Valor
C	1
ε	0.001
Kernel	Polinomial Grado 2, Polinomial Normalizado Grado 2 y RBF

En el segundo momento se experimentó con:

Parámetro	Valor
C	100
ε	0.001
Kernel	Polinomial Grado 2 y Polinomial Grado 3

Procesos Gaussianos.

Se utilizó la implementación de Weka para este modelo. Dicha implementación no considera el refinamiento de hyperparámetros. En los experimentos se utilizaron los tres *kernels* existentes en Weka. Los valores de los parámetros empleados se relacionan a continuación

Parámetro	Valor	
Kernel	RBF	$\sigma = 1.0$
	Polinomial	p = 2 y p=3
	Polinomial Normalizado	p = 2 y p=3
Ruido	1.0	

Regresión Lineal

Este modelo fue empleado utilizando tres configuraciones diferentes a partir del método de selección de los atributos. Estas configuraciones fueron:

1. Sin selección de atributos.
2. M5. Se recorren los atributos eliminando aquel que tenga menor coeficiente estandarizado hasta que no se observe mejora en la estimación del error dado por el criterio de información de Akaike.
3. Selección golosa.

Regresión lineal por Mínimos Cuadrados Medio (Least Median of Squares)
Se utilizó un tamaño de muestra de 1652

2.3 Conclusiones Parciales.

Al analizar los diferentes modelos propuestos para la experimentación se puede comprobar la amplia diversidad de configuraciones posibles a utilizar por lo que resulta

conveniente la división de los experimentos en dos momentos. Uno primero que determine la mejor configuración dentro de cada modelo y un segundo que permita establecer comparaciones entre los modelos como tal.

En la experimentación con el modelo de PSO se puede establecer la versatilidad en su empleo en la estimación de funciones continuas.

CAPÍTULO 3.

Análisis del comportamiento de los métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por creep en aceros ferríticos.

En éste capítulo se realiza un análisis de los resultados obtenidos en los experimentos enfatizándose en los preceptos teóricos que conducen a los mismos. Los datos de los resultados de los experimentos en ambos momentos se detallan en el ***Anexo 2***.

3.1 PSO en la estimación de funciones continuas con datos de *creep*.

Como fue descrito en el *Capítulo 2* del presente informe, PSO fue utilizado en tres roles a partir de la implementación del algoritmo clásico y una variante propuesta para el entrenamiento de redes neuronales conocida como APSO.

Entrenamiento de un MLP

Al analizar el uso de PSO en el entrenamiento del MLP se pueden observar los siguientes resultados:

➲ Al utilizar PSO1 se obtuvo un valor de *rmse* = 21.39. La implementación de APSO tuvo como hipótesis la idea de mejorar el rendimiento de PSO1 en cuanto a valor de *rmse*, teniendo en cuenta los valores obtenidos por *BackPropagation*. Se pretendía así, dilucidar la conjetura de que la superficie del error se caracterizaba por grandes llanuras seguidas de pasos empinados (***Figura 6***, epígrafe *1.2.3*), lo cual unido a la alta dimensionalidad del espacio de búsqueda podría estar provocando que regiones donde se encontraban las soluciones no fueran lo suficientemente exploradas. En el experimento se mantuvo fijo el número de iteraciones (20000) para poder realizar comparaciones. Se obtuvo un valor de *rmse* = 23.45 que como se observa no mejora el resultado de PSO1. De lo anterior se puede concluir uno ó ambos de los siguientes aspectos:

- una superficie de error, mejor que la lograda hasta el momento (en cuanto a exploración), no tiene las características planteadas, por tanto las modificaciones que impone APSO son anuladas en esta situación ó
- APSO muestra una convergencia más lenta hacia el mínimo de la función de aptitud, es decir, para llegar al mismo resultado que PSO1, se requiere de un mayor número de iteraciones. Un resultado con estas características fue observado en [43].

➲ De forma general, a pesar de que obtuvo el mejor resultado, PSO1 mostró una lenta convergencia (se necesitó de 20000 iteraciones para alcanzar el valor de *rmse* = 21.39) al ser comparado con *BackPropagation*, algoritmo clásico de entrenamiento de un MLP. Las consideraciones al respecto serán analizadas en el siguiente epígrafe para aprovechar un mejor contexto.

Optimización de parámetros de un MLP

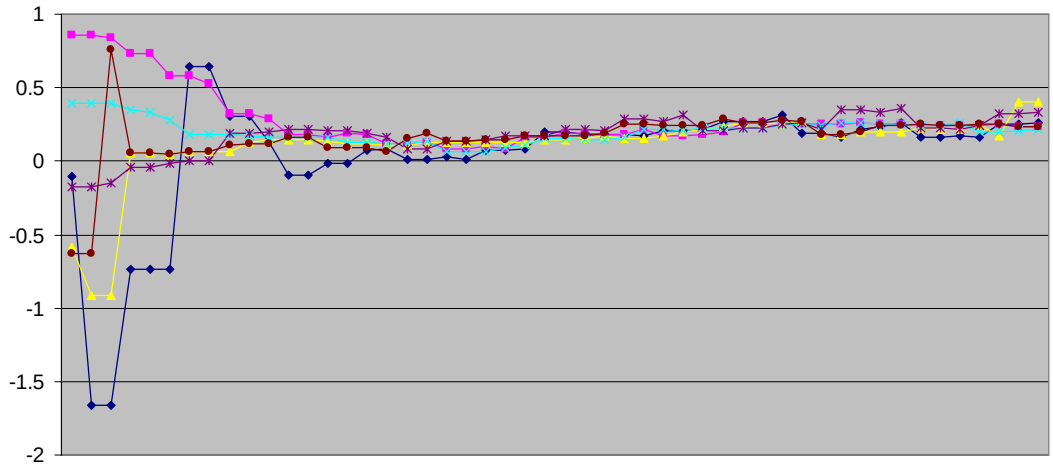
Un segundo enfoque en la utilización de PSO fue como optimizador del conjunto de parámetros que caracterizan a un MLP. En este caso, los parámetros para el algoritmo PSO1 fueron inicializados con valores pequeños por dos cuestiones: primero, la dimensión de las partículas es solo 2 y segundo, la concepción de la función de aptitud evidentemente genera un gran consumo de tiempo (se necesita entrenar y probar 50 x 20 = 1000 configuraciones de redes). La implementación como clasificador dentro de WEKA se justifica por el hecho de que en procesos como este, donde se realiza una búsqueda a ciegas de la configuración de un modelo (digamos el caso en que no se tiene ningún conocimiento de experto sobre el comportamiento de dicho modelo en los datos que se traten), PSO puede contribuir con un criterio de búsqueda rápida, lo cual es posible gracias a que PSO es independiente al dominio del problema que se trate. De esta misma forma, el clasificador presentado puede ser utilizado en vinculación con cualquier clasificador existente en WEKA, solo es necesario hacer los ajustes descritos en el *Capítulo 2*.

Regresión Lineal

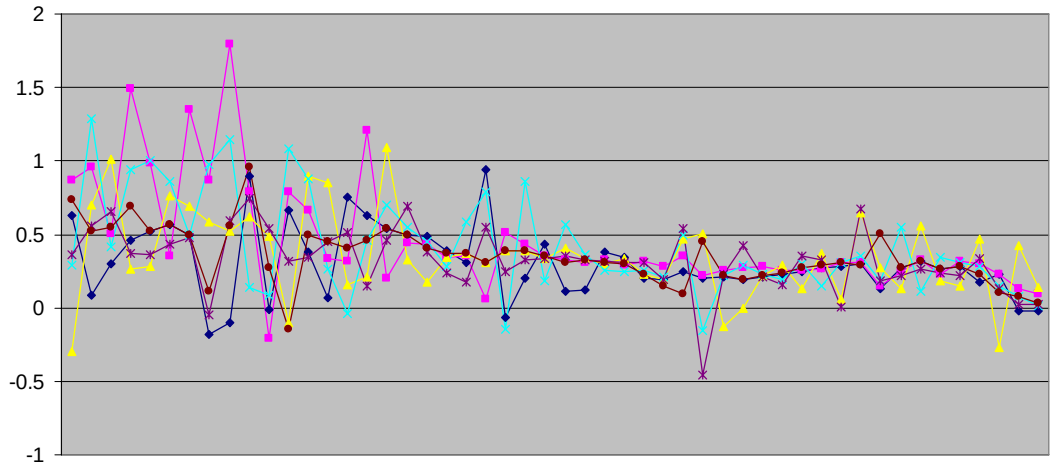
Al utilizar PSO para obtener una regresión lineal en la problemática del *creep*, se comparó el modelo clásico con PSO1. El resultado se describe a continuación:

➲ Introducir en el algoritmo PSO1, el elemento propuesto en MPPSO de reinicializar el vector de velocidad de las partículas con cierta frecuencia, induce mejoras en el comportamiento de PSO. En el caso de PSO clásico se obtuvo un valor medio de *rmse* = 102.53 mientras que al utilizar PSO1, el valor obtenido fue de 32.13. Este valor es similar al mejor valor obtenido por los restantes modelos de regresión lineal (bien establecidos por demás) utilizados en los experimentos (*rmse* = 31.91). El objetivo de dicha propuesta es contribuir a que la búsqueda en el espacio de soluciones sea disgregada frecuentemente

evitando que todo el enjambre sea conducido a un mínimo local. En el ***Gráfico 2*** se visualiza la trayectoria que siguen las partículas (se grafica una sola dimensión) en PSO clásico y PSO1. El eje de las ***x*** representa las iteraciones mientras que el de las ***y*** representa los valores que toma la dimensión.



*a) Trayectorias de 5 partículas en el algoritmo clásico de PSO (**rmse = 102.53**)*



*b) Trayectorias de 5 partículas en PSO1 (**rmse = 32.13**)*

***Gráfico 2**. Comparación del comportamiento de las trayectorias de 5 partículas. La disgregación mayor de la posición de las partículas en PSO1, por reinicialización frecuente del vector de velocidad, conduce a mejores valores de **rmse**.*

3.2 Análisis de los resultados de los diferentes métodos.

En este epígrafe se realiza un análisis de los resultados obtenidos por los diferentes métodos empleados en los experimentos.

En la ***Tabla 3*** se muestra una selección (las configuraciones básicas de mejor comportamiento) de los resultados dentro de cada método en el primer momento del experimento. Se refleja el promedio de la raíz del error cuadrático medio (*rmse*) obtenido a partir de las 30 corridas. En aquellos métodos donde se experimentó con diferentes configuraciones, los resultados aparecen ordenados según el ranking obtenido del test de Friedman. También se muestra si existe o no significación estadística al comparar par a par los resultados de las diferentes configuraciones dentro de cada método, según el procesamiento estadístico descrito en el *epígrafe 2.2.3*.

Nro	Método – Configuración	*rmse* promedio	Ranks según Friedman	Diferencia Significativa según test de Wilcoxon * existe – no existe		
				I	II	III
I	1 – b (P. Gauss. K-Pol)	16.34	1.00			
II	1 – a (P. Gauss. K-RBF.)	25.71	2.00	*		
III	1 – c (P. Gauss. K-Pol.Nor)	30.12	3.00	*	*	
	2 (Regresión Isotónica)	72.76				
	3 (LMS regression)	33.09				
I	4 – c (Reg. Lineal. Golosa)	31.93	1.93			
II	4 – b (Reg. Lineal. M5)	31.94	2.11	–		
III	4 – a (Reg. Lineal. No Sel)	31.94	2.13	–	–	
I	5 – e (Pace Reg AIC)	31.92	1.70			
II	5 – d (Pace Reg PACE6)	31.94	2.08	–		
III	5 – a (Pace Reg Emp.Bay)	31.95	2.23	*	–	
	6 (PLS)	31.91				
I	7 – (RBF 205 clusters)	26.13	1.80			
II	7 – (RBF 210 clusters)	26.52	1.87	–		
III	7 – (RBF 200 clusters)	27.23	2.33	*	–	
I	8 – b (SVMR K-Pol. G2)	15.11	1.00			
II	8 – a (SVMR K-Pol.N G2)	18.57	2.00	*		
III	8 – c (SVMR K-RBF)	34.57	3.00	*	*	
I	9 – (MLP 11 nodos ocultos)	18.98	2.08			
II	9 – (MLP 19 nodos ocultos)	19.09	2.93	*		
III	9 – (MLP 10 nodos ocultos)	19.13	3.18	*	–	
IV	9 – (MLP 18 nodos ocultos)	19.14	3.38	*	–	–

Tabla 3. Resultados de las configuraciones dentro de cada método. *Método-Configuración* se refiere a la numeración dada a las diferentes configuraciones en la *Tabla 2*, *epígrafe 2.2.1*

Como se puede observar los mejores resultados se obtuvieron para las configuraciones de SVMR, Procesos Gaussianos y MLP en ese orden. En el caso de los dos primeros, el *kernel* de mejores resultados fue el kernel polinomial de grado 2. Estos métodos fueron los seleccionados para realizar el segundo momento del experimento.

La ***Tabla 4*** muestra los resultados de los experimentos en el segundo momento.

	Método – Configuración	*rmse* promedio	Ranks según Friedman	Diferencia Significativa según test de Wilcoxon * existe – no existe					
				I	II	III	IV	V	VI
I	SVMR KPol,G=3,*C* = 100	12.47	1.10						
II	Proc Gauss, KPol, G=3	13.02	1.90	*					
III	SVMR KPol,G=2,*C* = 100	14.95	3.33	*	*				
IV	SVMR KPol, G2, C=1	15.11	3.73	*	*	*			
V	Proc Gauss, KPol, G2	16.34	4.93	*	*	*	*		
VI	MLP 11 nodos (BP)	18.59	6.00	*	*	*	*	*	
VII	MLP 11 nodos (PSO)	21.39	7.00	*	*	*	*	*	*

Tabla 4. *Resultados de los métodos de mejor comportamiento.*

En un primer análisis de los resultados se observa que las redes neuronales muestran resultados satisfactorios respecto a los métodos de regresión lineal (el mejor valor de *rmse* en los modelos de regresión lineal fue de 31.91, mientras que el MLP obtuvo 18.59). Este resultado es de esperarse si se tiene en cuenta que el MLP permite captar casi arbitrariamente cualquier relación no lineal entre las variables de entrada debido a la flexibilidad que brindan las funciones de los nodos de la capa oculta. Este modelo realiza un tipo de regresión multidimensional y no lineal, en la que no es necesaria la suposición inicial de una determinada forma funcional para el ajuste (lo cual es necesario en las regresiones lineales); en este sentido se suele apreciar en su aplicación práctica que supera a la regresión u otras técnicas lineales convencionales para espacios de dimensión alta (mayor que tres) [54]. Estos aspectos han sido corroborados por otros estudios rigurosos [55].

En el entrenamiento del MLP, la utilización del método clásico *Backpropagation* obtuvo mejores resultados al compararlo con PSO1 configurado para realizar un máximo de 20000 iteraciones (*rmse*=18.59 vs *rmse*=21.39). Este resultado no coincide con el obtenido en varios trabajos revisados en la literatura [37-41] donde la utilización de PSO en el entrenamiento de redes neuronales con una dimensionalidad (cantidad de pesos y umbrales que deben ser ajustados) no grande, ha demostrado ser un método eficaz que resuelve algunos problemas presentados por *BackPropagation* como es la lenta convergencia. En este caso, el resultado es motivado por la gran dimensionalidad que debe asumir PSO (430 parámetros libres en el caso del MLP con 11 nodos en la capa oculta). Lo anterior, unido a que la función de aptitud combina el ajuste de todas las dimensiones (en este caso, los pesos y umbrales del MLP) hace que las partículas no tengan forma de conocer que dimensiones influyen o no en su rendimiento haciendo que la convergencia sea extremadamente lenta. Esta problemática fue encontrada igualmente en el trabajo de *Dean Tsou* y *Cara MacNish* [43]. En el entrenamiento mediante *BackPropagation*, la minimización del error se lleva a cabo mediante descenso por el gradiente, a través de un algoritmo bien establecido ajustando los pesos directamente en la misma medida en que influyen en el error.

Al comparar los resultados del modelo de red neuronal de mejores resultados (MLP con 11 neuronas en la capa oculta y entrenado con *BackPropagation*) con el método SVMR se puede observar que este último presenta un mejor comportamiento. Los motivos de este resultado pueden estar relacionados con el dilema varianza – sesgo, presente tanto en estimadores paramétricos como no paramétricos. El MLP realiza estimaciones denominadas de modelo libre pues no se impone al problema un determinado modelo de partida, como por ejemplo, la línea recta en el método convencional de ajuste por mínimos cuadrados. Para que una red neuronal realice un ajuste óptimo, el número de ejemplos de entrenamiento debería tender a infinito [30], pues para un conjunto pequeño, suele ser muy sensible a casos particulares de pares entrada-salida seleccionados para realizar el aprendizaje. La causa es que la red neuronal, estimador de modelo libre, posee inherentemente una gran varianza, es decir, puede implementar muchísimos diferentes *mappings*, pero solamente implementará el correcto si se utiliza un conjunto de entrenamiento de tamaño idealmente infinito. En la literatura [56] se ha corroborado una

forma de inducir la cantidad óptima de ejemplos de entrenamiento en dependencia del orden del error de generalización (ε) con que se desee estimar y la cantidad de parámetros libres a ajustar en la red (w). Se plantea que debe estar en el orden de w/ε, en este caso, para ε=0.1 (un 10%) y la topología de mejor comportamiento (37 nodos de entrada, 11 nodos en la capa oculta y un nodo de salida) que contiene 430 parámetros libres entre pesos y umbrales serían necesario 4300 (430 * 10) ejemplos de entrenamiento para realizar un ajuste óptimo. La única forma de controlar la elevada varianza que la red neuronal posee inicialmente es introducir en su arquitectura algún tipo de sesgo o información apriorística sobre el problema a resolver. Este problema puede ser abordado con técnicas de regularización que mediante la introducción del sesgo, provocan que el *mapping* que implementa la red neuronal sea suave, es decir, que a entradas similares haga corresponder resultados próximos. Este sesgo puede ser introducido en la función de costo en forma de términos adicionales $\varphi(w_\alpha)$ que miden la desviación de los resultados actuales respecto de la restricción planteada

$$E[w_\alpha] = \frac{1}{2}\sum_{\mu}\sum_{k}\left[t_k^{\mu} - F_k^{\mu}(x)\right]^2 + \lambda\sum_{\alpha}\varphi(w_\alpha)$$

Muchos algoritmos como las redes de regularización minimizan una expresión de riesgo similar a la siguiente:

$$R[f] = R_{emp}[f] + \frac{\lambda}{2}\| w \|^2 \qquad (6)$$

siendo λ el denominado parámetro de regularización, que controla el compromiso entre el grado de suavidad de la solución frente al nivel de ajuste de los datos de entrenamiento que alcanza el modelo [57]. Estas características de regularización, están implícitas en la concepción del modelo de SVMR que se basa en el principio de minimización del riesgo estructural al introducir en la función de costo un término de control de capacidad ($||w||^2$), véase la función a minimizar (5) descrita en el epígrafe 1.2.4; esto hace esperar un mejor resultado. Minimizar (5) es equivalente a minimizar (6) si asumimos que $C = 1/(\lambda m)$. Para corroborar esta idea, basta observar las dos cuestiones siguientes.

a. el término $||w||^2$ constituye un control que regula la suavidad de la solución no permitiendo que esta se base solamente en el nivel de ajuste de los datos de entrenamiento lo cual pudiera traer como consecuencia el problema de sobre ajuste (*overfitting*) de parámetros si no se cuenta con ejemplos suficientes.

b. el parámetro *C* se concibe como un balance entre la llanura de la función a estimar y la cantidad hasta la cual se toleran errores mayores que ***ε*** (véase la descripción de la función de pérdida en el epígrafe 1.2.4)

Teniendo en cuenta el análisis anterior podemos fundamentar los resultados obtenidos en los experimentos. Se observa que el mejor valor de *rmse* obtenido por MLP fue de 18.59 para el caso de una topología con 11 nodos ocultos, este valor resulta significativamente diferente (según análisis estadístico, ***Tabla 5***) al obtenido por SVMR para el kernel polinomial de grado 2 y C = 1 que fue de 15.11 (evidentemente, un mejor valor).

			MLP (11 nodos) BP - SVMR Kernel Pol Grado 2 C=1
Z			-4.782(a)
Asymp. Sig. (2-tailed)			.000
Monte Carlo Sig. (2-tailed)	Sig.		.000
	99% Confidence Interval	Lower Bound	.000
		Upper Bound	.000
Monte Carlo Sig. (1-tailed)	Sig.		.000
	99% Confidence Interval	Lower Bound	.000
		Upper Bound	.000

Tabla 5. *Resultados del Test de Wilcoxon con técnica de Monte Carlo para establecer la significación de la diferencia entre los resultados del MLP entrenado con BackPropagation y SVMR con Kernel Polinomial grado 2 y C = 1. Como se observa la significación es menor que 0.05.*

Este resultado permite concluir:

➲ Puede existir un cierto nivel de sobre ajuste en el entrenamiento de la red, hipótesis probable si se analiza además el hecho de no contar con una cantidad óptima de ejemplos de entrenamiento para alcanzar niveles de errores bajos (como fue descrito anteriormente). Este elemento no puede ser resuelto por el modelo de MLP pues no tiene implementada ninguna técnica de regularización ocurriendo lo contrario con SVMR que desde su concepción, si la incluye. SVMR, mediante el elemento de control de capacidad, hace que ejemplos con posible ruido no influyan en la estimación de la función final y por tanto permite obtener mejores resultados.

➲ Existe una relación funcional polinomial entre las variables predictoras y la tensión de ruptura por *creep* (variable a estimar).

En SVMR, al aumentar el valor de C desde 1 hasta 100, estamos permitiendo que la función a estimar sea menos llana y exista una tolerancia menor a los errores mayores que ***ε*** (en los experimentos se consideró 0.001). Como se puede observar se logra una mejoría en el valor de *rmse* = 14.95 (muestra diferencia significativa, ***Tabla 6***). Esto reafirma el mejor comportamiento de SVMR al compararlo con el MLP en la problemática del *creep* pues realiza aproximaciones más exactas sin llegar a permitir sobreajuste.

			SVMR Kernel Pol Grado 2 C=1 - SVMR Kernel Pol Grado 2 C=100
Z			-2.396(a)
Asymp. Sig. (2-tailed)			.017
Monte Carlo Sig. (2-tailed)	Sig.		.015
	99% Confidence Interval	Lower Bound	.012
		Upper Bound	.018
Monte Carlo Sig. (1-tailed)	Sig.		.009
	99% Confidence Interval	Lower Bound	.006
		Upper Bound	.011

***Tabla 6**. Resultados del Test de Wilcoxon con técnica de Monte Carlo para establecer la significación de la diferencia entre los resultados de SVMR con Kernel Polinomial grado 2 variando C desde 1 hasta 100. Como se observa la significación es menor que 0.05.*

Un resultado aún mejor se obtuvo al utilizar en SVMR, el *kernel* polinomial con grado 3 cuyo *rmse* fue de 12.47 obteniéndose una diferencia significativa según el test de Wilcoxon (***Tabla 7***) incluso al compararlo con el resultado del *kernel* polinomial de grado 2 y C=100.

			SVMR Kernel Pol Grado 2 C=100 – SVMR Kernel Pol Grado 3 C=100
Z			-4.782(a)
Asymp. Sig. (2-tailed)			.000
Monte Carlo Sig. (2-tailed)	Sig.		.000
	99% Confidence Interval	Lower Bound	.000
		Upper Bound	.000
Monte Carlo Sig. (1-tailed)	Sig.		.000
	99% Confidence Interval	Lower Bound	.000
		Upper Bound	.000

***Tabla 7**. Resultados del Test de Wilcoxon con técnica de Monte Carlo para establecer la significación de la diferencia entre los resultados de SVMR con Kernel Polinomial grado 2 y C=100 respecto a SVMR con Kernel Polinomial grado 3 y C=100. Como se observa la significación es menor que 0.05.*

Un aspecto interesante en el análisis de los resultados es el siguiente. Si se observa el ***Gráfico 1*** del epígrafe 2.1 se puede apreciar que la distribución de ejemplos según el particionamiento establecido para los valores de la variable a estimar, no es uniforme. Existe, a partir de las particiones intermedias, una tendencia a disminuir considerablemente la cantidad de ejemplos disponibles, lo cual tiene una implicación directa en el hecho de que no se pueda identificar o "aprender" adecuadamente, relaciones entre variables que conduzcan a los valores de *creep* que poseen dichos ejemplos. Esta afirmación se puede comprobar al analizar visualmente la distribución del promedio de los errores según la partición de valores de *tensión de ruptura por creep* propuesta en el ***Gráfico 2***. Como se observa, los mayores promedios de errores se encuentran en las particiones que contienen poca cantidad de ejemplos.

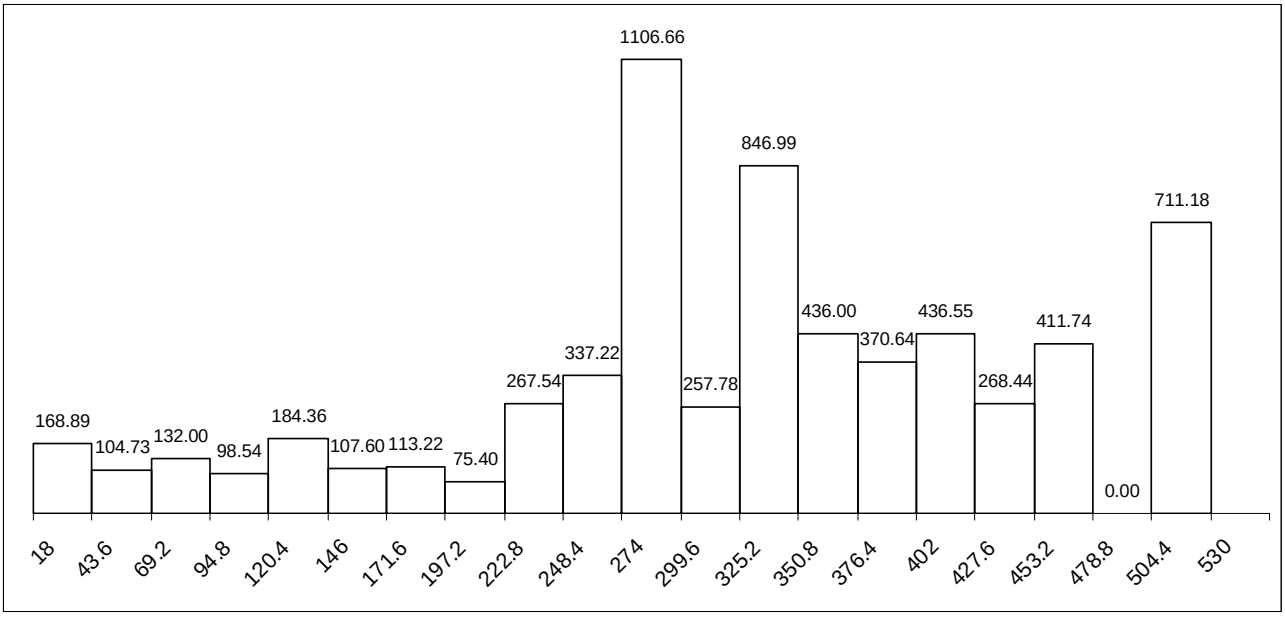


Gráfico 2. *Distribución del error en el pronóstico en base a los valores de tensión de ruptura por creep (el eje X representa los valores de tensión y el eje Y el promedio del error obtenido en el pronóstico de los ejemplos con valores comprendidos en el rango representado por el eje X)*

El análisis anterior hace concluir que en la utilización de modelos de estimación de funciones continuas para el pronóstico de la tensión de ruptura por *creep* con vistas a ser empleado en el diseño de nuevas aleaciones, no es suficiente con medir el error promedio obtenido por dicho modelo sino también poder contar con un estimado de la incertidumbre con que se pronostican los nuevos valores. Esto incrementa la importancia de los resultados obtenidos con el modelo de Procesos Gaussianos que permite obtener dicha incertidumbre a partir de determinar la distribución de probabilidad de un conjunto de

funciones que permiten hacer estimaciones de nuevos valores y además muestra un comportamiento eficiente (*rmse* = 13.02).

3.3 Comparación con resultados descritos en la literatura.

En la literatura se han descrito trabajos de estimación de la tensión de ruptura por *creep* en aceros ferríticos utilizando redes neuronales [28, 29]. En el trabajo de *F. Brun* y colectivo de autores [28] se empleó un MLP entrenado en un ambiente bayesiano a partir del mismo conjunto de datos utilizado en el presente trabajo. Esto nos permite realizar cierta comparación entre los resultados obtenidos aunque en dicho trabajo no aparece explícitamente el indicador raíz del error cuadrático medio como medida del error de prueba. En este caso se utilizó como medida del error, el valor calculado como:

$$e = \frac{1}{2}\left(\sum_{\mu=1}^{m}\left(t_\mu - y_\mu\right)^2\right),$$

donde ***m*** es la cantidad de casos de prueba, $\boldsymbol{y_\mu}$ y $\boldsymbol{t_\mu}$ constituyen el valor estimado por la red neuronal y el valor experimental respectivamente para el ejemplo $\boldsymbol{\mu}$.

Según las gráficas expuestas en dicho trabajo, el mejor valor obtenido del indicador ***e*** fue de 0.154 lo cual significa un valor de raíz del error cuadrático medio (*rmse*) de 16.57 si consideramos *m* = 1033, utilizamos la forma de normalización descrita en dicho trabajo y el cálculo de *rmse* se realiza según la fórmula descrita en el *epígrafe 2.2.2* de la presente investigación.

El método descrito al ser desarrollado en un ambiente bayesiano, tiene la posibilidad de realizar las estimaciones indicando barras de error que indican la incertidumbre de la predicción.

Aunque un análisis comparativo de este resultado con el obtenido por Procesos Gaussianos (*rmse* = 13.02) y SVMR (*rmse* = 12.74) no está bien fundamentado desde el punto de vista estadístico ni experimental pues no se cuenta con los elementos suficientes y exactos, sí constituye un resultado que justifica la posibilidad de ver a estos modelos como métodos de buenas prestaciones en comparación con las redes neuronales en la estimación de la tensión de ruptura por *creep*.

3.4 Conclusiones parciales.

Los resultados obtenidos mantienen plena consistencia con los preceptos teóricos que fundamentan los diferentes modelos de estimación de funciones continuas utilizados. Se destacan los resultados obtenidos por SVMR y Procesos Gaussianos, modelos de reciente creación y que tienen influencias tanto del campo del aprendizaje estadístico como del aprendizaje automatizado. Específicamente se pueden concluir los siguientes aspectos:

- La modificación de PSO conocida como APSO, no induce mejoras en el comportamiento del modelo cuando es utilizado como método de entrenamiento de un MLP para estimar la tensión de ruptura por *creep* en aceros ferríticos a partir de las variables descritas en la presente investigación.
- En el modelo de PSO, la reinicialización frecuente del vector velocidad, constituye un elemento eficaz para evitar caer en mínimos locales.
- Al comparar *Backpropagation* con PSO como método de entrenamiento de un MLP se puede observar que éste último muestra resultados menos eficientes cuando la dimensionalidad es alta. Este resultado se debe a que PSO actualiza las dimensiones en las partículas (pesos de la red) de forma global sin tener en cuenta cual influye realmente en la mejora de la solución a través del proceso de entrenamiento.
- El MLP mostró mejores resultados que todos los modelos de regresión lineal lo cual se justifica por su flexibilidad y ser un método que realiza estimaciones de modelo libre. Esto contrasta con los modelos lineales a los que se les impone una forma funcional lineal desde su concepción.
- El método de SVMR obtuvo resultados satisfactorios al compararlo con el MLP. Las causas están relacionadas con las ventajas inherentes que posee el principio de minimización del riesgo estructural en que se basa SVMR cuando se compara con el principio de minimización del riesgo empírico que fundamenta al MLP.
- Los datos de *creep* muestran ciertas relaciones polinomiales entre las variables predictoras y la tensión de ruptura por *creep* lo cual fue corroborado al obtener los mejores resultados utilizando un kernel polinomial en SVMR.

- Los Procesos Gaussianos constituyen un método eficaz para el pronóstico de la tensión de ruptura por *creep* pues, además de mostrar estimados de error satisfactorios en la predicción de nuevos casos, permite obtener un nivel de incertidumbre para dicha predicción.

CONCLUSIONES

A partir de los resultados obtenidos en la presente investigación se pueden establecer las siguientes conclusiones:

1. Modelos de estimación de funciones continuas de reciente desarrollo dentro del campo del aprendizaje automatizado como el caso de SVMR y Procesos Gaussianos, muestran un mejor comportamiento que los modelos tradicionales en el pronóstico de la Tensión de Ruptura por *Creep.*
2. El uso del principio de Minimización del Riesgo Estructural conduce a resultados más eficientes que los obtenidos por el principio de Minimización del Riesgo Empírico en la estimación de funciones continuas con datos de *creep.*
3. En el pronóstico de la tensión de ruptura por c*reep* es importante establecer la incertidumbre de la predicción y en este sentido los Procesos Gaussianos pueden constituir un modelo de gran prestación y eficiencia.
4. Los datos de *creep* muestran relaciones polinomiales entre las variables predictoras y la tensión de ruptura por *creep*.
5. El modelo de optimización PSO, puede ser empleado con versatilidad en la estimación de funciones continuas.
6. En el empleo de PSO para el entrenamiento de un MLP, la velocidad de convergencia hacia un valor óptimo de la función de aptitud, es susceptible a la dimensionalidad de las partículas; en este caso, la cantidad de parámetros libres a ajustar en la red.

RECOMENDACIONES

La investigación realizada constituye un estudio inicial sobre el comportamiento de métodos de estimación de funciones continuas en el pronóstico de la tensión de ruptura por *creep* y por tanto resulta un punto de partida en el estudio de esta problemática. Se recomienda:

1. Utilizar el Capítulo 3 del presente informe como base teórica para futuras investigaciones sobre el pronóstico de la tensión de ruptura por *creep* y el diseño de softwares con este fin.
2. Utilizar el modelo de Procesos Gaussianos como base de un software que permita obtener el pronóstico de la tensión de ruptura por *creep*.
3. Estudiar la posible vinculación de PSO con técnicas de visualización a fin de mejorar su rendimiento en cuanto a velocidad de convergencia para problemas que impliquen elevada dimensionalidad en las partículas.
4. Utilizar los resultados expuestos como criterios de partida para el estudio del comportamiento de los métodos de funciones continuas aquí descritos, sobre otras características de los materiales.

REFERENCIAS BIBLIOGRÁFICAS

1. H. K. D. H. Bhadeshia y T. Sourmail: *Japan Society for the Promotion of Science, Committee on Heat–Resisting Materials and Alloys*, 2003, 44, 299–314.
2. H. K. D. H. Bhadeshia: ISIJ International, 1999, 39, 966–979.
3. H. K. D. H. Bhadeshia: ISIJ International, 2001, 41, 626–640.
4. T. G. Langdon: *Dislocations and creep, Dislocations and Properties of Real Materials*, Institute of Metals, London, 1985, 221–238.
5. B. Wilshire: *Strength, Fracture and Complexity 1*, 2003, 39–46.
6. R. G. Thiessen: *Mathematical Modelling of Weld Phenomena 7*, published by T.U. Graz, Austria, 2003.
7. A. Kumar, W. Zhang, C.-H. Kim and T. DebRoy: *Mathematical Modelling of Weld Phenomena 7*, published by T.U. Graz, Austria, 2003.
8. M. A. Yescas–Gonzalez, H. K. D. H. Bhadeshia and D. J. C. MacKay: *Materials Science and Engineering A 311,* 2001, 162–173.
9. Y. F. Yin and R. G. Faulkner: *6th International Charles Parsons Turbine Conference, Engineering Issues in Turbine Machinery, Power Plant and Renewables*, eds A. Strang, R. D. Conroy, W. M. Banks, M. Blackler, J. Leggett, G. M. McColvin, S. Simplson, M. Smith, F. Starr and R. W. Vanstone, Institute of Materials, London, 2003, 525–535.
10. H. K. D. H. Bhadeshia, A. Strang and D. J. Gooch: *International Materials Reviews*, 1998, 43, 45–69.
11. D. Venugopalan, J. S. Kirkaldy: *Hardenability Concepts with Applications to Steels*, eds D. V. Doane and J. S.Kirkaldy, TMS–AIME, Warrendale, Pennsylvania, USA, 1978, 249–267.
12. F. Abe: *Fourth International Conference on Recrystallisation and Related Phenomena*, eds T. Sakai and H. G. Suzuki, The Japan Institute of Metals, 1999, 289–294.
13. S. Yamasaki and H. K. D. H. Bhadeshia: *Materials Science and Technology*, 2003, 19, 723–731.

14. N. Fujita and H. K. D. H. Bhadeshia: *Materials Science and Technology*, 2001, 17, 403–408.

15. J. D. Robson, H. K. D. H. Bhadeshia: *Materials Science and Technology*, 1997, 13, 631–639.

16. S. J. Jones, H. K. D. H. Bhadeshia: *Acta Materialia*, 1997, 45, 2911–2920.

17. S. J. Jones, H. K. D. H. Bhadeshia: *Metallurgical and Materials Transactions A 28A*, 1997, 2005–2013.

18. Y. F. Yin and R. G. Faulkner: *Materials Science and Engineering A A344*, 2003, 92–102.

19. D. E. Coates: *Metallurgical Transactions*, 1972, 3, 1203–1212.

20. D. E. Coates: *Metallurgical Transactions*, 1973, 4, 395–396.

21. D. E. Coates: *Metallurgical Transactions*, 1973, 4, 1077.

22. H. K. D. H. Bhadeshia: *Progress in Materials Science*, 1985, 29, 321–386.

23. D. R. F. West y N. Saunders: *Ternary Phase Diagrams in Materials Science*, 3rd edition, Institute of Materials, London, 2002.

24. M. Evans: *Materials Science and Technology*, 1999, 5, 647–658.

25. R. W. Evans y B. Wilshire: *Creep of Metals and Alloys*, The Institute of Metals, London, 1985.

26. D.J.C. MacKay: *Bayesian Methods for Neural Networks: Theory and Applications*, Neural Networks Summer School, Cambridge University, U.K., 1995.

27. T. Sourmail, H. K. D. H. Bhadeshia y D. J. C. MacKay: *Materials Science and Technology*, 2002, 18, 655–663.

28. F. Brun, T. Yoshida, J. D. Robson, V. Narayan, H. K. D. H. Bhadeshia, y D. J. C. MacKay: *Materials Science and Technology*, 1999, 15, (5), 547-554.

29. D. Cole, C. Martin – Moran, A.G. Sheard, H. K. D. H. Bhadeshia y D.J.C MacKay: *Science and Technology of Welding and Joining*, 2000, 5, (2), 81-89.

30. B.M. del Brío, A.S. Molina: *Redes Neuronales y Sistemas Difusos*, ed. Alfaomega Ra-Ma, 2001, Zaragoza, Universidad de Zaragoza, 2001, 41-79.

31. Lee, S. *Supervised learning with Gaussian potentials.* En [32] pp. 189-227, 1992.

32. Kosko, B. *Neural Networks for Signal Processing.* Prentice-Hall,1992.

33. Tou, J. T., Gonzhlez, R. C. *Pattern Recognition Principles.* Addison-Wesley Pub. Comp., 1974.

34. Wasserman, P. D. *Advanced Methods in Neural Computing.* Van Nostrand Reinhold, 1993.

35. Hush, D. R, Home, B. G. *Progress in supervised neural networks. What's new since Lipprnann?.* IEEE Signal F'roc. Mag., 8-38, enero 1993.

36. J. Kennedy y R. C. Eberhart: *Particle Swarm Optimization*, Proc. IEEE International Conference on Neural Networks, 1995, 1942-1948.

37. El-Gallad, A. I., El-Hawary, M., Sallam, A. A. & Kalas, A. 2001: *Swarm-intelligently trained neural network for power transformer protection*, Canadian Conference on Electrical and Computer Engineering, 2001, Vol. 1, 265-269.

38. Mendes, R., Cortez, P., Rocha, M. & Neves, J.: *Particle swarm for feedforward neural network training*, Proceedings of the 2002 International Joint Conference on Neural Networks, 2002, Vol. 2, 1895-1899.

39. F. Van den Bergh y A.P. Engelbrecht: *Cooperative Learning in Neural Networks using Particle Swarm Optimizers*, South AfricanComputer Journal, 2000, 26, 84-90.

40. R.C. Eberhart, Y. Shi: *Evolving Artificial Neural Networks*, Proceedings of the International Conference on Neural Networks and Brain, Beijing, China, 1998, 5-13.

41. V.G. Gudise, G.K. Venayagamoorthy: *Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks*, Proceedings of the Swarm Intelligent Symposium, Indianapolis, USA, 2003, 110-117.

42. Buthainah Al-kazemi y Chilukuri K. Mohan: *Training Feedforward Neural Networks using Multi-Phase Particle Swarm Optimization*, Proc. Congress on Evolutionary Computation, Honolulu, Hawaii, 2002.

43. Dean Tsou y Cara MacNish: *Adaptive Particle Swarm Optimization for High-Dimensional Highly Convex Space Search.* School of Computer Science & Software Engineering. University of Western Australia.

44. V. Vaptnik, V.N.: *The nature of Statical Learning Theory*, New York: Wiley, 1995.

45. V. Vaptnik, S. Golowich, A. Smola: *Support vector method for function approximation, regression, estimation, and signal processing*, In M. Mozer, M. Jordan, and T. Petsche, editors, Advances in Neural Information Processing Systems 9, Cambridge MA: Mit Press, 1997, 281- 287.

46. Li- Na Li, Chao- Zhen Hou: *The Identification of Industrial Processes Based on SVM*, Proceedings of the First International Conference on Machine Learning and Cybernetics, Beijin, 2002.

47. XunkaiWei, Yinghong Li, ChenWang, Jianming Lu: *A Novel Identification Model of Aeroengine Based on Support Vector Machines*, Proceedings of the 5th World Congress on Intelligent Control and Automation, 2004, 200- 203.

48. Ming Guang,WeiWu Yan y Zhan Ting Yuan: *Study of Nonlinear Identification Based on Support Vector Machine*, Proceedings of 3th International Conference on Machine Learning and Cybernetics, 2004, 3287- 3290.

49. A.J. Smola y B.Schölkopf: *A tutorial on Support Vector Regression*, Neuro COLT2 Technical Report Series, 1998.

50. J.C. Platt: *Fast Training of Support Vector Machines Using Sequential Minimum Optimization*, in Schölkopf, Burges and Smola, Eds., Advances in Kernel Methods—Support Vector Learning, Cambridge MA: MIT Press, 1998, 185-208.

51. S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy: *Improvements to the SMO Algorithm for SVM Regression*, In: IEEE Transactions on Neural Networks, 1999.

52. David J. C. Mackay (1998). Introduction to Gaussian Processes. Dept. of Physics, Cambridge University, UK.

53. C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006.

54. Hecht-Nielsen, R.: *Neurocomputing*, Addison Wesley, 1990.

55. Principe, J. C., Euliano, N. R., Lefebvre, W. C.: *Neural and Adaptive Systems. Fundamentals Through Simulations*. John Wiley, 2000.

56. Haykin, S.: *Neural Neworks. A Comprehensive Foundation,* 2da edición, Prentice-Hall, 1994, 1999.

57. Poggio, T., Girosi, F.: *Networks for approximation and learning,* Proc. of the IEEE, 1990, 1481-1497.

Anexo 1. *Kernels implementados en WEKA*

En WEKA hasta la *versión 3.5.5* están implementados los siguientes *Kernels*

1. Kernel de Funciones de Base Radial *(RBF Kernel)*

$$K(x,y) = \exp^{-(\sigma * < x-y, x-y >^2)}$$

2. Kernel Polinomial (*PolyKernel*)

$$K(x,y) = < x, y >^p \quad \text{ó} \quad K(x,y) = (< x, y > +1)^p$$

Donde *p* es el orden del polinomio

3. Kernel Polinomial Normalizado

$$K(x,y) = \frac{< x, y >}{\sqrt{< x, x >< y, y >}}$$

donde, $< x, y >$ = *PolyKernel* (x, y)

Anexo 2. *Resultados de los Experimentos*

Resultados de los modelos utilizados en el primer momento del experimento.

Modelos lineales.

Datos	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
Data01	74.54	33.98	32.78	32.79	32.94	32.97	99.02	32.86	32.85	32.83	32.95
Data02	77.21	34.22	33.03	32.71	32.81	32.66	114.49	33.06	32.93	32.95	32.66
Data03	75.40	33.61	32.19	32.15	32.07	32.19	123.34	32.09	32.20	32.16	32.03
Data04	73.04	34.49	33.71	33.78	33.67	33.85	91.00	33.67	33.80	33.76	33.64
Data05	68.99	31.34	30.15	30.16	30.14	30.16	93.70	30.18	30.03	29.90	30.10
Data06	75.20	35.94	34.38	34.49	34.56	34.44	103.64	34.56	34.31	34.55	34.31
Data07	70.59	34.08	32.67	32.60	32.63	32.77	89.03	32.65	32.73	32.81	32.55
Data08	67.96	33.54	33.02	33.04	33.02	33.45	95.81	33.07	32.98	33.01	32.72
Data09	70.72	31.22	30.86	30.99	30.85	31.30	122.31	30.87	31.03	30.94	30.88
Data10	69.87	33.01	31.63	31.60	31.56	31.62	84.66	31.52	31.73	31.61	31.77
Data11	79.00	37.79	35.52	35.52	35.55	35.41	121.10	35.53	35.52	35.58	35.55
Data12	72.28	34.82	32.65	32.61	32.65	32.82	91.28	32.65	32.69	32.66	32.62
Data13	70.29	31.59	31.62	31.68	31.55	31.74	119.89	31.51	31.77	31.66	31.65
Data14	73.80	33.19	31.47	31.32	31.47	31.34	121.37	31.45	31.44	31.42	31.55
Data15	73.60	31.85	31.49	31.50	31.51	31.50	88.54	31.49	31.42	31.47	31.51
Data16	75.02	32.55	31.55	31.36	31.55	31.44	94.18	31.58	31.41	31.50	31.44
Data17	67.36	28.04	28.02	28.11	28.05	28.77	101.69	27.99	28.11	28.03	28.19
Data18	70.32	31.53	30.62	30.55	30.51	30.62	94.11	30.49	30.31	30.46	30.60
Data19	75.68	33.69	32.13	32.15	32.10	32.19	91.56	32.08	32.14	32.22	32.16
Data20	72.40	31.60	30.92	30.88	30.82	32.89	115.37	30.82	31.03	30.90	30.86
Data21	73.56	33.25	32.92	32.96	32.85	33.06	108.01	32.82	33.03	32.93	32.89
Data22	77.94	34.00	32.13	32.08	32.12	32.13	97.16	32.12	32.08	32.16	32.01
Data23	69.94	32.65	31.48	31.40	31.53	31.31	112.54	31.53	31.34	31.39	31.29
Data24	74.79	33.06	32.00	32.00	32.00	32.22	107.97	31.97	31.93	31.93	32.12
Data25	76.79	34.04	31.62	31.72	31.65	31.60	93.61	31.61	31.68	31.80	31.46
Data26	68.43	29.74	29.60	29.59	29.46	29.96	110.12	29.46	29.50	29.52	29.41
Data27	72.70	33.75	32.44	32.52	32.48	33.01	83.46	32.37	32.39	32.56	32.44
Data28	71.64	32.77	31.64	31.57	31.65	32.02	96.84	31.63	31.67	31.65	31.55
Data29	73.96	34.10	32.72	32.96	32.75	32.83	124.71	32.70	33.02	32.85	32.87
Data30	69.72	33.39	31.30	31.36	31.33	31.48	85.45	31.31	31.24	31.24	31.39
rmse promedio	***72.76***	***33.09***	***31.94***	***31.94***	***31.93***	***32.13***	***102.53***	***31.92***	***31.94***	***31.95***	***31.91***
Desv. St.	3.07	1.81	1.41	1.41	1.43	1.33	13.12	1.44	1.43	1.45	1.39

Leyenda

Columna	Modelo
C1	Regresión Isotónica
C2	Regresión Lineal por Mínimos Cuadrados Medios (LMS)
C3	Regresión Lineal. Selección de atributos M5.
C4	Regresión Lineal. Sin selección de atributos.
C5	Regresión Lineal. Selección de atributos golosa.
C6	Regresión Lineal con PSO1.
C7	Regresión Lineal con PSO clásico.
C8	*Pace Regresión*. Estimador AIC
C9	*Pace Regresión*. Estimador Pace6
C10	*Pace Regresión*. Estimador *Empirical bayes*.
C11	PLS (*Partial Least Squares*)

Modelos no Lineales

Datos	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
Data01	18.18	18.42	18.33	18.26	22.27	22.78	27.68	14.75	19.92	36.05	16.42	27.33	31.41
Data02	19.44	19.64	19.70	19.76	22.93	22.71	23.90	16.13	19.15	37.35	17.26	28.83	32.03
Data03	18.40	18.34	18.80	19.10	29.21	29.15	33.63	15.73	17.35	36.67	14.78	25.73	30.86
Data04	18.82	19.12	19.14	19.41	32.59	32.25	28.05	14.87	18.95	35.57	16.19	27.28	32.14
Data05	17.42	17.66	18.00	18.03	21.37	21.12	25.25	13.43	16.11	31.50	14.59	23.47	27.86
Data06	19.45	19.76	19.54	20.02	29.29	34.08	31.75	15.04	19.63	37.62	15.94	27.05	32.97
Data07	18.89	19.29	19.38	19.38	26.41	26.20	27.38	14.01	17.75	34.16	16.12	24.50	30.17
Data08	19.19	19.20	19.25	19.31	23.51	27.91	30.39	16.75	19.90	34.85	17.91	26.88	31.33
Data09	19.23	19.07	18.63	18.61	26.49	26.96	25.24	13.83	17.69	32.16	15.32	24.36	28.16
Data10	19.09	19.33	19.24	19.08	29.99	30.39	28.22	15.71	19.32	34.29	16.75	25.62	29.97
Data11	18.53	18.47	18.38	18.54	24.45	24.18	27.26	14.38	19.77	41.29	16.07	29.50	35.17
Data12	18.64	19.04	18.90	18.79	24.24	23.91	24.15	16.54	19.32	34.09	18.01	26.30	29.94
Data13	18.46	18.36	19.27	18.32	25.30	25.83	26.85	13.40	18.35	32.80	15.92	26.22	30.42
Data14	19.90	20.54	20.05	19.90	23.77	24.71	24.22	15.95	19.11	34.95	17.14	25.28	29.08
Data15	20.60	20.64	20.51	20.55	27.41	29.25	28.59	16.88	19.45	32.73	17.83	26.14	29.21
Data16	18.42	18.53	18.44	18.42	28.99	30.35	29.51	14.23	17.85	33.90	16.00	23.12	29.28
Data17	17.06	16.95	17.14	17.29	23.24	23.41	23.62	12.54	15.73	28.05	13.86	21.69	25.68
Data18	18.04	18.03	17.93	18.31	28.41	28.43	29.08	15.16	17.22	32.48	16.31	23.54	27.90
Data19	19.58	19.70	19.57	19.88	27.29	27.30	29.73	15.58	18.43	36.05	16.14	26.53	31.44
Data20	20.30	20.13	20.06	20.42	29.35	30.09	29.47	16.10	19.49	33.15	17.00	25.68	29.51
Data21	20.06	20.16	20.18	20.20	22.60	22.54	22.50	14.30	19.12	34.07	16.54	24.74	30.26
Data22	19.84	19.86	19.92	20.02	31.06	30.63	30.66	15.72	18.82	36.30	16.63	26.46	30.85
Data23	16.68	16.72	17.25	16.87	19.95	19.60	20.14	13.88	17.09	33.58	14.68	25.55	28.46
Data24	19.63	19.44	19.51	19.66	34.48	31.72	32.79	15.70	18.17	35.48	16.84	27.86	31.60
Data25	19.37	19.47	19.33	19.08	26.15	26.06	26.04	14.82	17.49	35.19	15.53	25.45	29.61
Data26	17.95	17.73	17.85	17.84	22.87	23.91	26.26	15.64	19.25	31.45	16.68	24.00	27.77
Data27	20.70	20.79	20.78	20.79	26.03	25.79	27.07	16.85	20.52	34.76	17.47	26.23	31.10
Data28	18.35	18.51	18.46	18.48	24.45	24.40	24.95	14.42	17.47	33.91	16.00	22.32	27.98
Data29	20.31	20.53	21.16	20.56	26.53	26.42	29.29	15.95	19.02	37.07	17.23	29.60	31.83
Data30	19.18	19.36	19.34	19.26	23.20	23.56	23.20	15.03	19.60	35.52	16.96	24.00	29.70
rmse promedio	***18.98***	***19.09***	***19.13***	***19.14***	***26.13***	***26.52***	***27.23***	***15.11***	***18.57***	***34.57***	***16.34***	***25.71***	***30.12***
Desv. St.	1.00	1.05	0.98	099	3.45	3.55	3.16	1.11	1.17	2.40	1.01	1.94	1.89

Leyenda

Columna	Modelo
C1	MLP 11 neuronas ocultas. Entrenamiento con *BackPropagation*
C2	MLP 19 neuronas ocultas. Entrenamiento con *BackPropagation*
C3	MLP 10 neuronas ocultas. Entrenamiento con *BackPropagation*
C4	MLP 18 neuronas ocultas. Entrenamiento con *BackPropagation*
C5	RBF con 205 clusters
C6	RBF con 210 clusters
C7	RBF con 200 clusters
C8	SVMR. Kernel Polinomial Grado 2 y C=1.0
C9	SVMR Kernel Polinomial Normalizado Grado 2 y C=1.0
C10	SVMR Kernel basado en RBF.
C11	Procesos Gaussianos. Kernel Polinomial Grado 2
C12	Procesos Gaussianos. Kernel basado en RBF.
C13	Procesos Gaussianos

Resultados de los modelos utilizados en el segundo momento del experimento.

Datos	C1	C2	C3	C4	C5	C6
Data01	17.58	21.11	23.20	15.42	11.82	12.28
Data02	19.12	23.42	22.35	16.08	13.45	14.06
Data03	18.00	18.89	22.81	15.61	13.10	13.20
Data04	18.58	20.54	24.82	15.09	12.49	13.00
Data05	17.15	18.84	22.44	13.06	10.89	11.67
Data06	19.04	21.12	24.06	14.94	12.75	13.43
Data07	18.32	21.91	24.35	13.61	12.68	12.71
Data08	19.00	21.25	24.85	16.92	12.53	13.80
Data09	18.50	21.23	22.03	13.17	11.23	11.89
Data10	18.81	20.89	21.42	15.48	12.62	13.08
Data11	17.94	21.69	23.42	14.08	13.09	12.37
Data12	18.22	20.94	23.05	16.11	13.12	14.13
Data13	17.99	21.06	26.58	14.00	11.89	11.87
Data14	19.55	21.35	23.57	15.54	12.75	13.36
Data15	20.18	22.78	25.78	16.52	13.38	14.26
Data16	17.90	21.41	23.51	13.81	11.74	12.40
Data17	16.42	18.99	22.55	12.76	10.38	11.19
Data18	17.93	19.84	22.96	15.05	13.24	13.73
Data19	19.17	20.43	24.62	15.64	12.38	12.90
Data20	19.98	29.32	22.16	15.35	12.63	13.90
Data21	19.53	22.44	25.17	14.44	11.67	12.81
Data22	19.42	22.25	22.79	15.87	12.69	13.19
Data23	16.65	19.26	21.62	13.68	11.27	11.86
Data24	19.41	21.58	23.89	14.89	12.80	14.12
Data25	18.96	20.82	20.92	14.56	12.12	12.99
Data26	17.57	20.17	22.35	15.31	12.62	13.14
Data27	20.15	22.48	27.40	16.45	12.80	13.69
Data28	17.99	21.11	20.45	14.79	12.32	13.03
Data29	19.72	22.65	25.00	15.62	15.33	13.58
Data30	18.83	22.00	23.24	14.61	12.42	13.11
rmse promedio	***18.59***	***21.39***	***23.45***	***14.95***	***12.47***	***13.02***
Desv. St.	0.98	1.90	1.61	1.07	0.91	0.80

Columna	Modelo
C1	MLP 11 neuronas ocultas. Entrenamiento con *BackPropagation*. 10000 iteraciones
C2	MLP 11 neuronas ocultas. Entrenamiento con PSO1 20000 iteraciones.
C3	MLP 11 neuronas ocultas. Entrenamiento con APSO 20000 iteraciones.
C4	SVMR. Kernel Polinomial Grado 2 y C=100
C5	SVMR. Kernel Polinomial Grado 3 y C=100
C6	Procesos Gaussianos. Kernel Polinomial Grado 3.