Modelos de estimación del software basados en técnicas de aprendizaje automático

María N. Moreno García y Francisco J. García Peñalvo

Universidad de Salamanca

Resumen: La realización de estimaciones fiables en las etapas iniciales del desarrollo de un proyecto es esencial para una efectiva toma de decisiones. Los métodos tradicionales de estimación son ampliamente conocidos y utilizados desde hace muchos años, sin embargo, actualmente están siendo reemplazados por modelos de predicción basados en técnicas de minería de datos cuya superioridad respecto a los primeros ha sido demostrada en numerosos estudios. En este capítulo se presenta una visión general de estos métodos, en particular de los algoritmos de aprendizaje automático, sus principales áreas de aplicación en el ámbito de la gestión de proyectos software y los procedimientos de validación de los mismos. Para ilustrar su uso se muestran varios casos de estudio: estimación del esfuerzo de desarrollo con redes neuronales, uso de un árbol de decisión para la estimación del tamaño del software y predicción de la calidad del software mediante programación genética y con multiclasificadores.

Palabras clave: Estimación de software, aprendizaje automático, minería de datos

Introducción

La estimación es una de las primeras y más importantes actividades de un proyecto. Consiste en predecir, en fases iniciales del ciclo de vida, características del software cuyo valor real sólo puede conocerse en etapas posteriores o cuando el proyecto ha finalizado. Las estimaciones condicionan la planificación y guían la toma de decisiones a lo largo de todo el proyecto, por lo que su fiabilidad tiene una gran repercusión tanto en la duración y coste del proyecto como en la calidad del software producido.

Las variables susceptibles de estimación son muchas. El tamaño, esfuerzo o coste del proyecto son las más habituales, aunque recientemente la estimación de características de calidad del software está adquiriendo gran importancia.

Estimar es una tarea complicada debido a la gran cantidad de factores (atributos del proceso o del producto) que influyen en el valor de la variable objeto de la predicción. Por ejemplo, el esfuerzo de desarrollo depende del tamaño y complejidad del producto, de la cualificación y experiencia del personal, de los métodos y herramientas utilizadas, etc. Para poder realizar predicciones sobre

características del proyecto actual es necesario disponer de información obtenida en proyectos pasados sobre las variables a estimar y los factores que les afectan. Estos datos permiten establecer un modelo de relaciones entre dichas variables que constituye la base del proceso de estimación. Los modelos de estimación clásicos más utilizados se basan en técnicas de regresión, cuyo principal inconveniente es la necesidad de formular una hipótesis sobre cuáles son los atributos más influyentes en la estimación de la variable objetivo y sobre el comportamiento de dichos atributos. La hipótesis generalmente viene dada en forma de una expresión matemática que relaciona los atributos con la variable a estimar. La expresión incluye varios parámetros cuyos valores se obtienen a partir de información histórica mediante el proceso de regresión. Las técnicas de aprendizaje automático no requieren la formulación de una hipótesis debido a que el análisis de datos tradicional orientado a la verificación es sustituido por un enfoque orientado al descubrimiento del conocimiento. Los algoritmos de aprendizaje utilizan los datos de proyectos pasados para inducir automáticamente un modelo que servirá para realizar estimaciones sobre nuevos proyectos.

Para situar dichas técnicas en el ámbito de la estimación del software, en este capítulo se presenta inicialmente una conocida taxonomía de los métodos de estimación de costes de un proyecto. Posteriormente se introducen las bases de los algoritmos de aprendizaje automático. El siguiente apartado recoge una breve revisión bibliográfica sobre el uso de estos métodos en el campo de la gestión de proyectos, seguida de varios ejemplos de aplicación de algunos de los métodos más representativos. Finalmente se exponen los procedimientos de validación de los modelos inducidos.

Clasificación de los métodos de estimación

La mayor parte de los métodos de estimación descritos en la bibliografía tienen como objetivo predecir el coste y esfuerzo del proyecto. Para clasificar estas técnicas pueden establecerse cuatro categorías (Fenton y Pfleeger, 1997):

1. Opinión de expertos. Un desarrollador o gestor describe los parámetros del proyecto y los expertos realizan estimaciones basadas en su experiencia.
2. Analogía. Enfoque más formal que la opinión de expertos. Los expertos comparan el proyecto propuesto con uno o más proyectos anteriores intentando encontrar similitudes y diferencias particulares.
3. Descomposición: Análisis minucioso de las características que afectan al coste del proyecto mediante la descomposición del producto o del proceso software. En el primer caso se realizan estimaciones individuales sobre los componentes en que se descompone el producto y en el segundo sobre las tareas de bajo nivel que forman parte de otras superiores. Las estimaciones de bajo nivel se pueden

combinar siguiendo diferentes procedimientos para producir una estimación global sobre el proyecto completo.

4. Modelos: Técnicas que identifican los factores clave que contribuyen al esfuerzo y generan un modelo matemático que relaciona dichos factores con el esfuerzo. Los modelos se basan normalmente en información obtenida de experiencias pasadas.

Los métodos del último grupo representan el enfoque más formal y son los que proporcionan resultados más fiables. En un principio recibieron el nombre de métodos algorítmicos o paramétricos debido que se utilizaban procedimientos como la regresión para producir un modelo constituido por una o varias expresiones matemáticas que relacionan el esfuerzo con una variable primaria, generalmente el tamaño, y varios factores de ajuste secundarios. El modelo COCOMO (Boehm, 1981) es el más representativo de este grupo.

Actualmente los métodos de estimación basados en modelos empíricos incorporan técnicas heurísticas consideradas no algorítmicas en el sentido de "no deterministas" (Dolado, 2000b), es decir, no sólo se considera la solución, sino la probabilidad de que esa solución sea cierta. La inducción de árboles de decisión, redes neuronales, algoritmos genéticos y otras técnicas de aprendizaje automático se están utilizando desde hace algunos años para construir modelos de predicción de diferentes atributos del software.

Fundamentos de los algoritmos de aprendizaje automático

Clasificación de las técnicas de minería de datos

Los algoritmos de aprendizaje automático se pueden englobar dentro de un grupo más general de técnicas denominadas de minería de datos, cuyo propósito es extraer conocimiento no explícito a partir de grandes volúmenes de datos. Las técnicas de minería de datos se pueden clasificar en dos grandes categorías: algoritmos supervisados o predictivos y algoritmos no supervisados o de descubrimiento del conocimiento (Weiss y Indurkhya, 1998). Los algoritmos de aprendizaje automático pertenecen a la primera categoría.

Los algoritmos supervisados o predictivos predicen el valor de un atributo (etiqueta) de un conjunto de datos, conocidos otros atributos (atributos descriptivos). A partir de datos cuya etiqueta se conoce se induce un modelo que relaciona dicha etiqueta y los atributos descriptivos. Esas relaciones sirven para realizar la predicción en datos cuya etiqueta es desconocida. Esta forma de trabajar se conoce como aprendizaje supervisado. En este grupo se encuentran, por una parte, algoritmos que resuelven problemas de clasificación debido a que trabajan con etiquetas discretas (árboles de decisión, tablas de decisión, inducción neuronal,

etc.) y, por otra, algoritmos que se utilizan en la predicción de valores continuos como son la regresión o las series temporales.

Los algoritmos no supervisados o de descubrimiento del conocimiento realizan tareas descriptivas como el descubrimiento de patrones y tendencias en los datos actuales (no utilizan datos históricos). El descubrimiento de esa información sirve para llevar a cabo acciones y obtener un beneficio científico o de negocio de ellas.

Otra clasificación que puede establecerse para los algoritmos de minería es la de métodos de caja negra (redes neuronales y estadística) y métodos orientados al conocimiento (árboles de decisión, reglas de asociación, reglas de decisión). Las técnicas de minería de datos abarcan ambos enfoques, aunque están más centradas en la segunda categoría. Por otra parte, en los últimos años se están desarrollando un conjunto de técnicas que se agrupan bajo el nombre de *soft computing* y que pueden aplicarse en el campo de la minería de datos. Su principal característica es la tolerancia a la imprecisión e incertidumbre, lo que les permite resolver problemas en entornos cambiantes, de forma robusta y a bajo coste (Jesús et al., 2004). Entre las más representativas se encuentran la lógica borrosa (*fuzzy*), los algoritmos evolutivos, algoritmos genéticos y redes neuronales (Tettamanzi y Tomassini, 2001).

El proceso de aprendizaje

Para llevar a cabo el aprendizaje supervisado se divide el conjunto de datos con etiqueta conocida en dos subconjuntos disjuntos, generalmente uno de mayor tamaño que el otro. El proceso se desarrolla en dos fases:

- Entrenamiento: construcción de un modelo de predicción usando el mayor de los subconjuntos.
- Prueba: se comprueba la validez del modelo inducido probándolo sobre el resto de los datos. El error se calcula comparando las predicciones realizadas por el modelo con el valor real del atributo etiqueta.

Cuando la etiqueta toma valores discretos, cada uno de estos valores representa una clase, por lo que el modelo predictivo resultante del proceso de aprendizaje se denomina clasificador.

El proceso de aprendizaje consiste en crear automáticamente un clasificador a partir de un conjunto de entrenamiento y de un inductor (figura 1):

- Conjunto de entrenamiento: datos utilizados para realizar el entrenamiento. Son datos cuya etiqueta se conoce, es decir, son ejemplos ya clasificados.
- Inductor: algoritmo que construye automáticamente un clasificador a partir de un conjunto de entrenamiento

El modelo inducido es el clasificador. Éste consiste en una serie de patrones que son útiles para distinguir las clases.

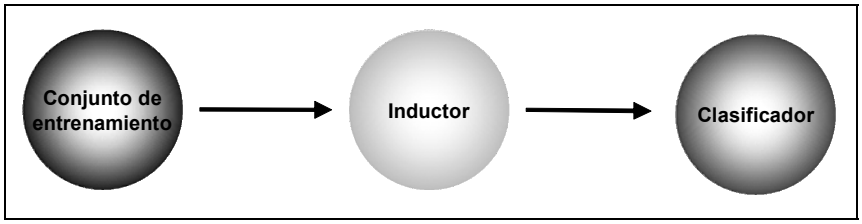

Fig. 1. Inducción de un clasificador

Una vez que se ha inducido el modelo se puede utilizar para predecir automáticamente la clase de otros registros no clasificados (figura 2).

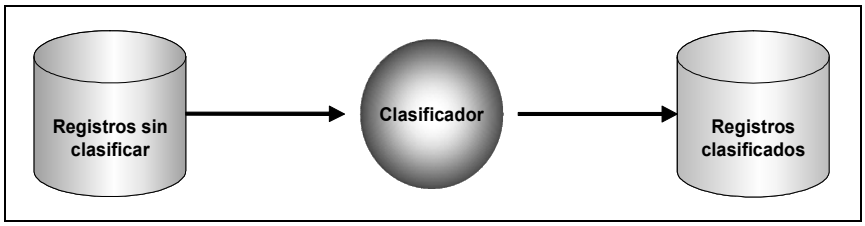

Fig. 2. Uso de un clasificador para predecir las clases de datos sin clasificar

Aplicación de la minería de datos en estimaciones de software

Las aplicaciones de las técnicas de minería de datos en el área de la Ingeniería del Software son múltiples (Mendonça y Sunderhaft, 1999). Se han utilizado tanto técnicas supervisadas como no supervisadas para resolver problemas de características variadas, aunque los algoritmos más utilizados han sido los de aprendizaje automático. En (Zhang y Tsai, 2003) se recogen diferentes aplicaciones de los mismos en la estimación de diferentes atributos del software. De ellos cabe destacar, por su influencia en el coste y calidad de los proyectos, los que se centran en la creación de modelos de estimación del tamaño del software, esfuerzo de desarrollo, esfuerzo de mantenimiento, defectos, facilidad de reutilización, facilidad de prueba, etc. Técnicas supervisadas como los algoritmos de inducción de árboles y tablas de decisión se han utilizado en la construcción de modelos de estimación del tamaño del software (Moreno y García 2005). Árboles de decisión y redes bayesianas se han usado también para la predicción de defectos del software (Fenton y Neil, 1999). El sistema propuesto permite derivar las distribuciones de probabilidad de defectos introducidos, detectados y densidad de defectos a partir de variables relativas a los procesos del ciclo de vida (especificación, diseño, implementación y prueba). Los árboles de decisión han sido también los modelos en

los que se han basado trabajos para la mejora de la fiabilidad (Tian y Palma, 1998) y estudios sobre la repercusión de algunas propiedades internas del software orientado a objetos (herencia, acoplamiento y complejidad) en su facilidad de reutilización (Mao et al., 1998).

El uso de redes neuronales también está ampliamente extendido, como ejemplo podemos citar su aplicación en la predicción de riesgos de mantenimiento en módulos de programa (Khoshgoftaar y Lanning, 1995) o su utilización en sistemas orientados a objetos para predecir el número de defectos en cada clase y el número de líneas modificadas por clase, usando como variables medidas relativas a la herencia, complejidad, acoplamiento, cohesión o asignación de memoria (Thwin y Quah, 2005). En (Dolado, 2000a) se realiza la validación de dos métodos de estimación del esfuerzo basados en técnicas de regresión. Los resultados de dichos métodos fueron mejorados mediante la aplicación de redes neuronales y programación genética.

Los modelos de predicción de la calidad del software se basan fundamentalmente en la detección de propiedades que conducen a defectos del software en diferentes etapas del desarrollo. En (Evett et al., 1998) se hace uso de técnicas de programación genética para desarrollar modelos de calidad que relacionan atributos medidos en fases iniciales del proyecto con defectos descubiertos en fases posteriores o durante el periodo de operación del sistema. La predicción del número de defectos de diseño es el objeto de otro estudio en el que se demuestra la superioridad de los clasificadores CBR (*Case Based Reasoning*) frente a las técnicas de regresión (Ganesan et al., 2000).

Recientemente se ha comprobado que la combinación de clasificadores mediante técnicas de *bagging*, *boosting* o *logitBoost* proporciona modelos de calidad del software más exactos y robustos que los producidos por los clasificadores individuales (Khoshgoftaar et al., 2003).

Aunque en menor grado, las técnicas no supervisadas o de descubrimiento del conocimiento también han contribuido a la mejora de la calidad del software. Este tipo de algoritmos suele utilizarse con fines descriptivos aunque es posible tomarlos como base para la construcción de clasificadores que serán usados para realizar predicciones. Así es posible realizar estimaciones del tamaño del software haciendo uso de un clasificador basado en reglas de asociación (Moreno et al., 2004b). El problema que presentan estos algoritmos es el gran número de reglas que generan. En (Moreno et al., 2004a) se ha propuesto un algoritmo de refinamiento de reglas de asociación que está particularmente indicado para el uso de las mismas con fines de clasificación. De la misma manera, las técnicas de *clustering* se han utilizado en la planificación del mantenimiento (Krohn y Boldyreff, 1999) y en la estimación de la fiabilidad del software (Podgurski et al., 1999). En (Dick et al., 2004) se realiza un estudio del uso de técnicas de minería de datos en el análisis de métricas del software. Valiéndose de un algoritmo de *fuzzy clustering* clasifican módulos con mayor o menor propensión a errores.

A continuación se muestran ejemplos de aplicación de algunas de estas técnicas. En el primer caso se utiliza una red neuronal para predecir el esfuerzo de desarrollo de software. El segundo ejemplo muestra un árbol de decisión como modelo de estimación del tamaño del software. Seguidamente se ilustra el uso de algoritmos genéticos en el descubrimiento de defectos en el software y, por último, se presenta un estudio en el que se demuestra la superioridad de varios multiclasificadores sobre los algoritmos individuales de aprendizaje automático en la predicción de la propensión de los módulos software a presentar defectos.

Redes neuronales artificiales

Las redes neuronales son colecciones de nodos conectados con entrada, salida y procesamiento en cada nodo. Entre las capas de entrada y salida visibles puede haber capas de procesamiento ocultas. Cada unidad de procesamiento en una capa está conectada con cada unidad de procesamiento de la siguiente capa. La fuerza de las relaciones se expresa mediante un factor de peso. Los factores se ajustan durante el entrenamiento de la red. Las salidas de la red representan los valores de las clases calculados a partir de los datos de entrada. El entrenamiento de la red se realiza con datos de proyectos pasados para los cuales se conoce el valor de la variable a estimar. Una vez inducido el modelo con dichos datos puede utilizarse para predecir el esfuerzo de proyectos en desarrollo.

El algoritmo de inducción neuronal más utilizado es el denominado de retropropagación (*back-propagation*) en el que los errores son propagados hacia atrás desde la capa de salida. Esta técnica es la que se utiliza en la estimación del esfuerzo de desarrollo de un proyecto en el siguiente ejemplo de aplicación (Srinivasan y Fisher, 1995). La arquitectura de la red se muestra en la figura 3. Como puede observarse, dispone únicamente de una capa oculta de procesamiento, un vector de entradas con diferentes atributos del software y una salida correspondiente al esfuerzo de desarrollo estimado. Las variables de entrada se derivan de las guías de coste del modelo COCOMO (Boehm, 1981), éstas son: AKDSI (instrucciones fuente entregadas ajustadas), TKDSI (instrucciones fuente entregadas totales), TURN (*Computer turnaround time*), TIME (restricciones de tiempo de ejecución), COB y PL1 (lenguajes de programación Cobol y PL1) y RVOL (volatilidad de los requisitos).

Las entradas y salidas están limitadas a valores numéricos, por lo que es necesario asignar valores numéricos a los atributos nominales. En el caso de estudio representado en la figura 3 cada valor (COB, PL1) del atributo nominal “lenguaje de programación” se corresponde con una entrada a la red. El valor 1 de dicha entrada representa la presencia del valor del atributo y el valor 0 representaría la ausencia del mismo.

Cada unidad de procesamiento de la red produce una salida que se calcula mediante la siguiente función lineal de las entradas:

$$1 / (1 + \exp [- (\sum_i w_i I_i)])$$

Donde ($\sum_i w_i I_i$) es la suma ponderada de las entradas, I_i, a un elemento de procesamiento.

Las salidas se van propagando desde la capa de entrada hasta la de salida final que produce la estimación del esfuerzo de desarrollo. Durante el entrenamiento de la red, si el valor no coincide con el real, los errores se propagan hacia atrás por la red guiando la modificación de los pesos de forma que se consigan reducir los errores. El proceso se repite hasta que se alcanza el criterio de convergencia establecido. Esto sucede después de un número especificado de iteraciones o cuando la diferencia entre el valor estimado y el valor real sea inferior a un valor de tolerancia. A veces la convergencia de las redes falla debido a impurezas en los datos o a la complejidad del problema. En el estudio presentado (Srinivasan y Fisher, 1995) se compararon los resultados conseguidos con los proporcionados por métodos clásicos obteniendo valores de MRE (magnitud del error relativo) del 70% para la red neuronal, 610% para el modelo COCOMO y 772% para el método SLIM.

Las redes neuronales tienen el inconveniente comentado anteriormente de aceptar únicamente valores numéricos como entradas y salidas, por lo que si se dispone de variables nominales es necesario recurrir a algún procedimiento de transformación en variables numéricas.

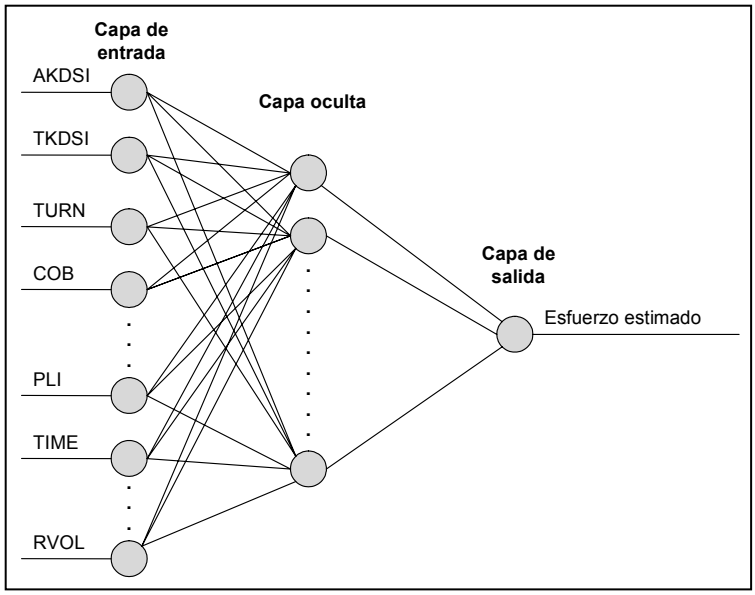

Fig. 3. Red neuronal para la estimación del esfuerzo de desarrollo

Árboles de decisión

La inducción de redes neuronales es una técnica bastante robusta, sin embargo, la lógica interna que sigue el modelo para realizar la clasificación no es fácilmente accesible. Este inconveniente no lo presentan los árboles de decisión. En ellos se muestran los valores de los atributos que proporcionan la separación de los datos en clases diferentes. El modelo presenta una estructura de árbol con *nodos* que simbolizan puntos de decisión y *hojas* correspondientes a puntos finales donde se recogen las observaciones de los datos.

La inducción de un árbol requiere un conjunto de datos de entrenamiento ya clasificados. En el ejemplo representado en la figura 4 se muestra un árbol de decisión que sirve para realizar estimaciones del tamaño del software en número de líneas de código (LOC) a partir de atributos que pueden obtenerse en fases iniciales del proyecto (Moreno et al. 2002). Para la representación se ha utilizado la herramienta *Mineset* de Silicon Graphics (Mineset, 1998).

El árbol se ha construido a partir de datos correspondientes a 42 proyectos (Dolado, 2000a) con información sobre los atributos propuestos por los métodos de estimación Mark II (Symons, 1991) y Verner and Tate (1992). Para formar el conjunto de entrenamiento se han considerado algunos de los atributos originales de esos métodos y otros derivados, aunque no todos aparecen en el modelo inducido.

A partir de los datos de entrenamiento se determina cuáles son los atributos que más influyen en la clasificación. Para ello se utiliza una medida basada en la cantidad esperada de información que proporciona el atributo. Ésta se obtiene calculando la entropía mediante la siguiente expresión:

$$\text{Entropía} = \text{I}\,(P(v_1), ..., P(v_n)) = \sum_{i=1}^{n} - P(v_i) \log_n P(v_i)$$

Siendo $P(v_i)$ la probabilidad de la clase iésima y n el número de clases posibles. La entropía es 1 cuando las probabilidades de las clases son iguales.

En este caso los atributos ordenados de mayor a menor influencia son los siguientes:

- RELATION: Número total de relaciones en el diagrama entidad-relación
- NTRNSMKII: Número de transacciones (definidas en el método Mark II)
- DATAELEMENT: Número total de elementos de datos
- NOC: Número de componentes (definidos en el método de Verter y Tate)

El atributo que mejor discrimina las clases se coloca en el nodo raíz del árbol. Sus valores sirven para dividir el conjunto de entrenamiento en subpoblaciones que pertenezcan a diferentes clases, creando de esta forma un nuevo nivel de la estructura jerárquica. Si no se consigue la clasificación completa de los datos, es decir, si existen subpoblaciones formadas por ejemplos (registros) de clases distintas, se recurre al siguiente atributo en orden de influencia, convirtiendo el nodo hoja del árbol en un nodo de decisión a partir del cual se realizará una nueva

partición. El proceso continúa hasta que no se consigan mejoras significativas en la clasificación o hasta que se alcance un número de niveles establecido.

Las barras que aparecen en cada nodo del árbol de la figura 4 simbolizan los registros. Su color indica la clase a la que pertenecen y su altura el peso de los registros de la clase correspondiente. El primer aspecto importante que se aprecia es que el atributo RELATION colocado en la raíz del árbol sirve para separar los datos en dos grupos bien diferenciados. Los registros con valores de ese atributo superiores a 88.5 pertenecen en su totalidad a alguna de las tres clases correspondientes a proyectos de mayor tamaño mientras que el resto de los registros pertenecen a las clases de menor número de líneas de código.

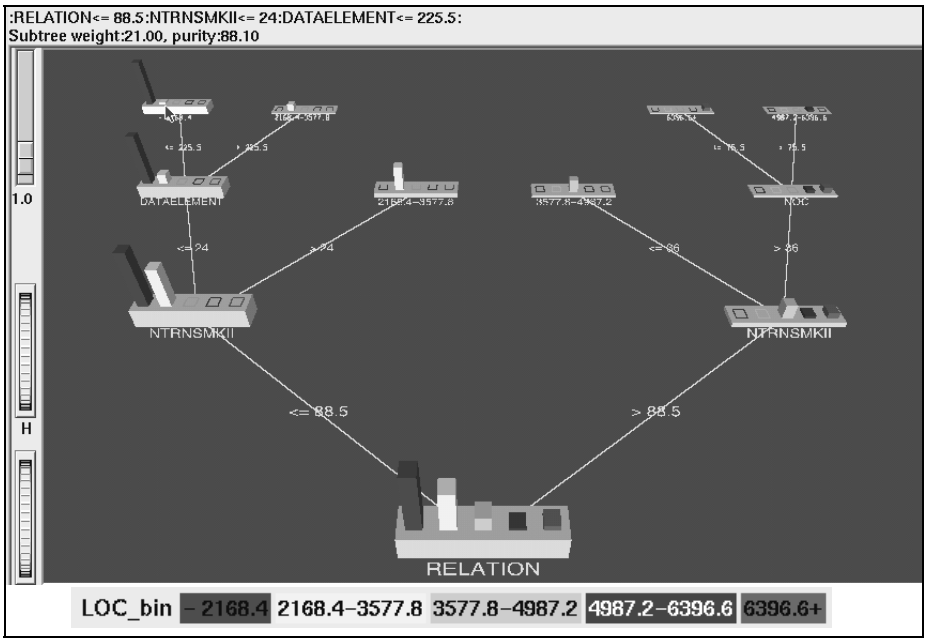

Fig. 4. Árbol de decisión para estimar el tamaño del software

En las hojas del árbol aparecen ejemplos que comparten características y que tienen cierta probabilidad de pertenecer a una clase. La probabilidad viene dada por el peso de los registros de dicha clase en el nodo hoja. En el caso ideal, en cada hoja del árbol deberían aparecer registros de una sola clase, lo que indicaría que el modelo permite discernir totalmente las clases en función de los valores de los atributos que aparecen en los nodos de decisión. Cuando la probabilidad de que todos los registros de un nodo pertenezcan a una sola clase es del 100% la entropía tiene un valor de 0, tomando el valor de 1 cuando las probabilidades de todas las clases son iguales, es decir, cuando en el nodo existe el mismo número de registros de cada clase. Para valorar la clasificación en un nodo se puede aplicar una medida

denominada pureza (1 - entropía), cuyo valor es del 100% cuando en el nodo hay únicamente registros de una clase. En el ejemplo de la figura 4 se puede observar que la pureza es del 100% en todos los nodos hoja excepto en el señalado por el cursor que es de un 88.1%.

Los ejemplos de cada nodo hoja comparten la misma regla de decisión, que no es más que la descripción del camino desde el nodo raíz al nodo hoja. El conjunto de todas las reglas de decisión del árbol conforman el clasificador que servirá para realizar predicciones sobre datos no clasificados. Las reglas obtenidas del árbol del ejemplo se recogen en la figura 5.

Para calcular los errores del modelo inducido se utilizó la técnica de validación cruzada (ver apartado de validación de los modelos). El error cuadrático medio variaba dependiendo del tamaño del conjunto de prueba y del número aleatorio elegido para realizar la partición de los datos en conjuntos de prueba y entrenamiento, pero en todos los casos se obtuvieron errores del 0% o próximos a esa cifra.

SI (RELATION > 88.5 **y** NTRNSMKII <= 36) **ENTONCES** LOC: 3577.8 – 4987.2

SI (RELATION > 88.5 **y** NTRNSMKII > 36 **y** NOC > 75.5) **ENTONCES** LOC: 4987.2 - 6386.6

SI (RELATION > 88.5 **y** NTRNSMKII > 36 **y** NOC <= 75.5) **ENTONCES** LOC > 6386.6

SI (RELATION <= 88.5 **y** (NTRNSMKII > 24 **ó** (NTRNSMKII <= 24 **y** DATAELEMENT > 225.5)))

ENTONCES LOC: 2168.4 - 3577.8

SI (RELATION <= 85.5 **y** NTRNSMKII <= 24 **y** DATAELEMENT <= 225.5)

ENTONCES LOC = 2168.4 - 3577.8 (probabilidad = 4.76%) **ó** LOC < 2168.4 (probabilidad = 95.24%)

Fig. 5. Reglas obtenidas del árbol de decisión

Programación genética

Los algoritmos genéticos se encuentran entre las técnicas más recientes de aprendizaje automático. Son algoritmos evolutivos basados en los conceptos de adaptación, supervivencia y reproducción de la teoría de Darwin. Los modelos producidos están formados por poblaciones de generaciones sucesivas de individuos. Los métodos clásicos representan a los miembros de la población como cadenas de dígitos binarios de longitud fija. Las posibles soluciones se van seleccionando entre los individuos mejor adaptados. Las soluciones buenas producen descendientes que sustituyen a otros peores, manteniendo muchas de las características de sus antecesores. Los miembros de nuevas generaciones se forman utilizando dos operadores genéticos: *cruce*, que combina características de dos individuos para crear nuevos individuos, y *mutación*, que opera cambiando aleatoriamente varios componentes de un individuo seleccionado. La calidad de la solución se estima mediante una función de ajuste que determina lo bueno que es un individuo dentro de la población en cada generación.

La programación genética es una extensión de los algoritmos genéticos clásicos en la que las estructuras que evolucionan no son secuencias de bits sino programas. Esta técnica es la que se ha utilizado en (Liu y Khoshgoftaar, 2001) para realizar predicciones sobre la calidad del software. El modelo les permite predecir qué módulos de software serán propensos a defectos y cuáles no, dados otros atributos resultantes de la aplicación de métricas de producto y de proceso en etapas tempranas del desarrollo. La función de ajuste se define en función del número y coste de los distintos tipos de clasificaciones incorrectas. Los errores de "tipo I" son los que se producen cuando el modelo predice que un módulo es propenso a errores y en realidad no lo es. El error inverso lo definen como de "tipo II" y es el que tiene un coste mayor ya que cuanto más tarde se descubra un defecto en el software más costosa resulta su corrección. El algoritmo de programación genética selecciona cinco individuos en cada iteración. Para seleccionar el mejor modelo producido en las sucesivas iteraciones del algoritmo se sigue el criterio de tomar aquél que produzca el índice más equilibrado entre los errores de tipo I y II, con el número de errores de tipo II más bajo posible. En los casos de estudio considerados esta técnica ha producido mejores resultados que otros métodos de regresión, como puede observarse en la tabla 1.

Errores	Método	
	Regresión	Programación Genética
Tipo I	30.60%	20.19%
Tipo II	32.18%	27.59%
Índice total de clasificaciones incorrectas	30.94%	21.78%

Tabla 1. Errores en la predicción de la propensión a defectos

Multiclasificadores

La precisión de los clasificadores puede variar en función del espacio de hipótesis con el que se trabaje. Una forma de mejorar la precisión de las predicciones consiste en crear multiclasificadores mediante la combinación de los modelos de clasificación obtenidos con diferentes métodos básicos. Otra ventaja adicional de estas técnicas es la reducción del problema de sobreadaptación (*overfitting*) que consiste en encontrar una regularidad en los datos propia del conjunto de entrenamiento y que no puede hacerse extensiva a otros datos.

Los multiclasificadores construyen un conjunto de hipótesis y combinan de alguna forma las predicciones del conjunto para clasificar los ejemplos. Se selecciona en cada caso la mejor predicción entre todas las ofrecidas por los diferentes modelos, siendo la votación mayoritaria el criterio de selección más simple y más utilizado. Existen muchos procedimientos para la creación de

multiclasificadores: *Bagging*, *boosting*, *logitBoost*, *co-learning*, *stacking*, *cascading*, etc. El método de *bagging* (Breiman, 1996) consiste en crear diferentes clasificadores con el mismo algoritmo pero con diferentes conjuntos de entrenamiento. Cada uno de éstos se crea por selección aleatoria y con reemplazamiento de una muestra de ejemplos del mismo tamaño que el conjunto de entrenamiento original. De entre todas las predicciones ofrecidas por los clasificadores se selecciona la que tenga mayor número de votos. *Boosting* (Freund y Schapire, 1996) se basa en la ponderación de los ejemplos. De forma iterativa se van construyendo modelos que minimicen los errores de los construidos previamente mediante la asignación de pesos mayores a los ejemplos clasificados incorrectamente en la iteración anterior. Además, cuando se utilizan para clasificar ejemplos reales, también se ponderan los modelos en función de su comportamiento en la fase de prueba. Una variación de esta técnica es el método de *logitBoost* (Friedman et al., 2000) en el que se utiliza un procedimiento de regresión para ajustar los modelos. La estrategia seguida en el método de *stacking* (Wolpert, 1992) es diferente, ya que éste es un método híbrido que genera diferentes clasificadores con diferentes algoritmos de aprendizaje. Puede utilizarse la votación mayoritaria para seleccionar la clase entre todas las que proporcionan los diferentes modelos o se puede utilizar un procedimiento de meta-aprendizaje consistente en crear un nuevo modelo que establezca la forma de combinar los modelos inducidos previamente (figura 6) (Hernández et al., 2004).

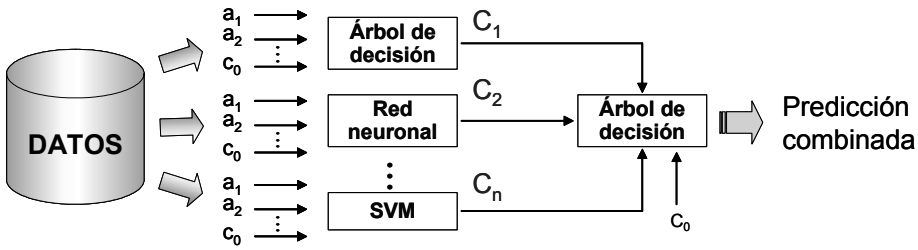

Fig. 6. Método de *stacking*

Tres de las técnicas comentadas anteriormente (*bagging*, *boosting* y *logitBoost*) se han aplicado en un estudio sobre calidad del software (Khoshgoftaar et al., 2003) y se han comparado con los algoritmos de inducción de árboles de decisión C4.5 (Quinlan, 1993) y *decision stump* (Witten y Frank, 2000). El objetivo fue clasificar módulos de software (ficheros de código fuente) en dos clases, propensos a defectos y no propensos. Los atributos descriptivos usados para construir los modelos fueron los siguientes:

- BASE_LOC: Número de líneas de código de la versión del fichero fuente antes de la fase de codificación. Representa código autogenerado.
- SYST_LOC: Número de líneas de código de la versión del fichero fuente entregado para las pruebas del sistema.

- BASE_COM: Número de líneas de código comentado de la versión del fichero fuente antes de la fase de codificación. Representa código autogenerado.
- SYST_COM: Número de líneas de código comentado de la versión del fichero fuente entregado para las pruebas del sistema.
- INSP: Número de veces que se inspeccionó el fichero fuente antes de las pruebas del sistema.

Los experimentos realizados mostraron que, en general, con los modelos combinados se alcanza mayor exactitud que con los clasificadores individuales. Asimismo se encontró que las técnicas de *boosting* proporcionan mejores resultados que las de *bagging*.

Validación de los modelos

Cuando se construye el clasificador se puede predecir cuál será su índice de error cuando se aplique a datos sin clasificar. Algunas de las técnicas que se pueden utilizar son:

– Validación simple (*holdout*)
– Validación cruzada
– *Bootstrap*
– Matrices de confusión, etc.

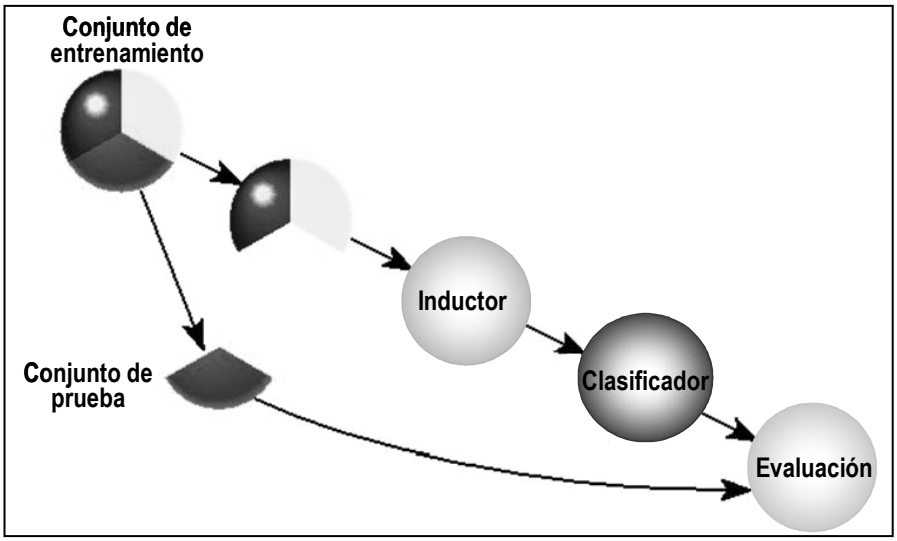

Fig. 7. Técnica *holdout* de estimación de errores

La validación simple o técnica *holdout* consiste en utilizar una porción de registros como conjunto de entrenamiento para construir el modelo y el resto como conjunto de prueba (figura 7). La precisión del modelo inducido vendrá dada por la relación entre el número de clasificaciones correctas obtenidas en la prueba y el número total de ejemplos del subconjunto de prueba.

En la validación cruzada (*cross-validation*) los datos se dividen en k subconjuntos mutuamente excluyentes del mismo tamaño aproximadamente (Dietterich, 1997). Se realizan k entrenamientos tomando en cada uno de ellos como conjunto de prueba un subconjunto diferente y construyendo el modelo con los subconjuntos restantes (figura 8). El índice de error estimado es la media de los errores obtenidos en cada entrenamiento. Se puede repetir t veces. En ese caso se construyen y evalúan k * t clasificadores.

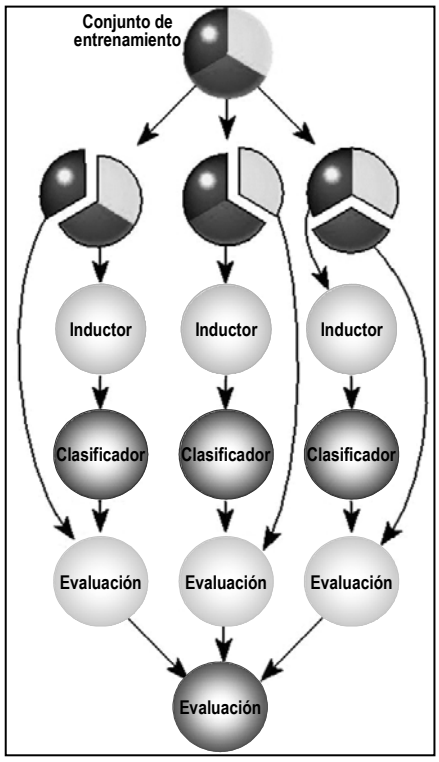

Fig. 8. Validación cruzada

Un método apropiado para estimar los errores de los modelos predictivos cuando se dispone de pocos datos es el de *Bootstrap*. En ella, para un conjunto con N ejemplos, el conjunto de entrenamiento se forma realizando un muestreo aleatorio con reposición de N ejemplos, es decir, pueden aparecer ejemplos repetidos. Los ejemplos no elegidos por la muestra forman el conjunto de prueba.

La técnica de las matrices de confusión, no sólo permite conocer el error del modelo predictivo sino que también muestran el tipo de las predicciones correctas e incorrectas cuando se aplica el modelo sobre el conjunto de prueba (Figura 9). Las predicciones correctas están representadas por las barras que aparecen sobre la diagonal, mientras que el resto de las barras indican el tipo de error cometido (qué valor ha predicho el modelo y cual es el valor verdadero). La altura de las barras es proporcional al peso de los registros que representan.

Cuando los errores llevan asociada una pérdida que puede cuantificarse es posible aplicar otras técnicas de validación como las matrices de pérdida, el análisis ROC (*Receiver Operating Characteristic*), las curvas ROI (*Return-On-Investment*), etc.

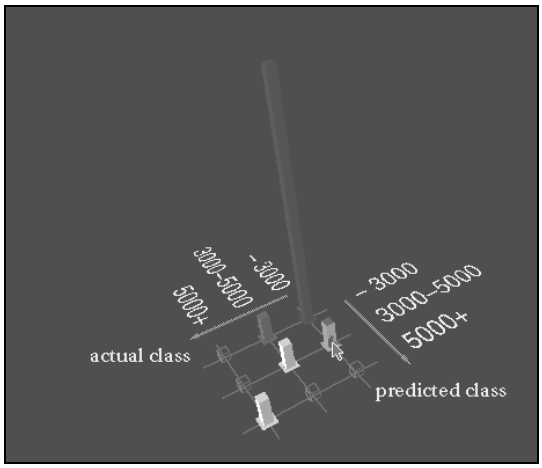

Fig. 9. Matriz de confusión

Conclusiones

Los métodos de estimación del software han sufrido una importante evolución en los últimos años. Se ha pasado de procedimientos tan poco formales como la opinión de expertos hasta complejos modelos matemáticos de estimación que relacionan múltiples variables. El salto más significativo ha sido el paso de los modelos de regresión a modelos inducidos con algoritmos de aprendizaje automático. La superioridad de estas técnicas se debe fundamentalmente a la generación automática de un modelo de predicción en el que aparecen los atributos más influyentes en la estimación de la variable objetivo sin necesidad de formular una hipótesis. La mayoría de los métodos clásicos de estimación usan un modelo paramétrico en forma de expresión matemática que calibran a partir de datos históricos, en cambio, los algo-

ritmos de aprendizaje automático proporcionan modelos no paramétricos sin que sea necesario realizar presunción alguna sobre la forma de la función que relaciona las variables implicadas en la estimación. En estos métodos, la hipótesis es el propio modelo inducido automáticamente a partir de los datos con el algoritmo de aprendizaje.

Por otra parte, los enfoques algorítmicos o paramétricos son generalmente inapropiados para modelar las complejas relaciones entre variables que existen en muchos entornos de desarrollo de software, por lo que producen resultados inexactos incluso habiendo realizado una calibración del modelo.

En la actualidad, la inducción de árboles de decisión, redes neuronales, algoritmos genéticos, multiclasificadores y otras técnicas de aprendizaje automático se están utilizando con éxito en la obtención de modelos de estimación de las más variadas características del software como tamaño, esfuerzo de desarrollo, esfuerzo de mantenimiento, defectos, facilidad de reutilización, facilidad de prueba, etc.

Los árboles de decisión tienen la cualidad de hacer explícitos los atributos relevantes en la predicción. Esto no sucede con otras técnicas como las redes neuronales en las que la lógica interna que sigue el modelo para realizar la clasificación no es fácilmente accesible, aunque estas últimas son más robustas debido a la ponderación de los datos. Los algoritmos englobados dentro de la computación flexible (*soft*) tienen una capacidad de adaptación que les permite obtener una solución aceptable a bajo coste para problemas que pueden formularse de forma imprecisa. Los multiclasificadores amplían el espacio de hipótesis mejorando la precisión y solventando algunos problemas que presentan los clasificadores individuales como puede ser la sobreadaptación a los datos de entrenamiento.

Referencias

Boehm BW (1981) Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ

Breiman L (1996) Bagging Predictors. Machine Learning, 24(2):123-146

Dick S, Meeks A, Last M, Bunke H, Kandel A (2004) Data Mining in Software Metrics Databases. Fuzzy Sets and Systems 145:81-110

Dietterich TG (1997) Machine Learning Research: Four Current Directions. The AI Magazine 18(1):97-136

Dolado JJ (2000a) A validation of the Component-Based Method for Software Size Estimation. IEEE Transactions on Software Engineering 26(10):1006-1021

Dolado JJ (2000b) Estimación del Tamaño y Coste de los Proyectos Software. En: Dolado JJ, Fernández L (coordinadores) Medición para la Gestión en la Ingeniería del Software. Ra-ma, Madrid, pp 209-224.

Evett M, Khoshgoftar T, Chien P, Allen E (1998) GP-Based Software Quality Prediction. En: Proc. of third Annual Genetic Programming Conference, pp 60-65

Fenton N, Neil M (1999) A Critique of Software Defect Prediction Models. IEEE Transaction on Software Engineering 25(5):675-689.

Fenton NE, Pfleeger SL (1997) Software Metrics. A Rigorous & Practical Approach. PWS Publishing Company, Boston

Freund Y, Schapire RE (1996) Experiments with a New Boosting Algorithm. En: Proc. 13th Int. Conference on Machine Learning, ICML'96, pp 148-156

Friedman J, Hastie T, Tibshirani R (2000) Additive Logistic Regression: A Statistical View of Boosting. The Annals of Statistics 28:337–407

Ganesan K, Khoshgoftar T, Allen E (2000) Cased-based Software Quality Prediction. International Journal of Software Engineering and Knowledge Engineering 10(2):139-152

Hernández J, Ramírez MJ, Ferri C (2004) Introducción a la Minería de Datos. Pearson Educación, Madrid

Jesús MJ, González, P, Herrera, F (2004) Extracción del conocimiento con algoritmos evolutivos y reglas difusas. En: Hernández J, Ramírez MJ, Ferri C, Introducción a la Minería de Datos. Pearson Educación, Madrid, pp 383-419

Khoshgoftaar TM, Lanning DL (1995) A Neural Network Approach for Early Detection of Program Modules having High Risk in The Maintenance Phase. J. Systems Software 29(1):85-91

Khoshgoftaar TM, Geleyn E, Nguyen L (2003) Empirical Case Studies of Combining Software Quality Classification Models. En: Proc. of Third International Conference on Quality Software (QSIC'03), Dallas, Texas, pp 40-51

Krohn U, Boldyreff C (1999) Application of Cluster Algorithms for Batching of Proposed Software Changes. J. Softw. Maint. Res. Pract. 11:151-165

Liu Y, Khoshgoftaar TM (2001) Genetic Programming Model for Software Quality Classification. En: Proc. of the Sixth IEEE International Symposium on High Assurance Systems Engineering, pp 127-136

Mao Y, Sahraoui H, Lounis, H (1998) Reusability Hypothesis Verification Using Machine Learning Techniques: A Case Study. En: Proc. of 13th IEEE Int. Conference on Automated Software Engineering

Mendonça, MG, Sunderhaft NL (1999) Mining Software Engineering Data: A Survey. Technical Report, DoD Data and Analysis Center for Software, DACS-SOAR-99-3

Mineset user's guide (1998) v. 007-3214-004, 5/98, Silicon Graphics

Moreno MN, Miguel LA, García FJ, Polo MJ (2002) Data mining approaches for early software size estimation. En: Proc. 3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02), Madrid, Spain, pp 361-368

Moreno MN, García FJ, Polo MJ (2004a) Mining Interesting Association Rules for Prediction in the Software Project Management Area. Lectures Notes in Computer Science 3181:341-350

Moreno MN, Miguel LA, García FJ, Polo MJ (2004b) Building Knowledge Discovery-Driven Models for Decision Support in Project Management. Decision Support Systems 38: 305-317

Moreno MN, García FJ (2005) Improving Estimations in Software Projects with Data Mining Techniques. Projects & Profits (Special Issue: Software Development Projects) 6:25-37

Podgurski A, Masri W, McCleese Y, y Wolff FG (1999) Estimation of Software Reliability by Stratified Sampling. ACM Trans.on Soft.Eng.and Methodology 8(3):263-283

Quinlan JR (1993) C4.5: Programs for Machine Learning. Morgan Kauffman

Srinivasan K, Fisher D (1995) Machine Learning Approaches to Estimating Software Development Effort. IEEE Transactions on Software Engineering 21(2):126-137

Symons CR (1991) Software Sizing and Estimating MKII FPA. John Wiley and Sons

Thwin MMT, Quah TS (2005) Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics. The Journal of Systems and Software 76(2005):147–156

Tettamanzi A, Tomassini M (2001) Soft Computing. Integrating Evolutionary, Neural and Fuzzy Systems.Springer

Tian J, Palma J (1998) Analyzing and Improving Reliability: A Tree-Based Approach. IEEE Software 15(2):97-104

Verner J, Tate G (1992) A Software Size Model. IEEE Transaction of Software Engineering 18(4):265-278

Weiss SM, Indurkhya N (1998) Predictive Data Mining. A Practical Guide. Morgan Kaufmann Publishers, San Francisco

Witten IH, Frank SE (2000) Data Mining. Morgan Kaufmann Publishers

Wolpert DH (1992) Stacked Generalization. Neural Networks 5:241-259