Trabajo Práctico Nro 2

Materia
Inteligencia Artificial

Alumnos : Luis Brassara
Siless Viviana

Informe

Dataset:

Este dataset tiene como atributos 16 distintas leyes que se han votado en el congreso de EEUU y la categoría se corresponde con la agrupación política a la que pertenecen los congresistas que votaron. Cada congresista puede votar por si o por no y puede ser demócrata o republicano. En total hay 435 instancias de las cuales 267 son demócratas y 168 republicanos.

Se puede observar en el dataset que la mayoría de las leyes son votadas en bloque, es decir, un congresista suele votar lo mismo que los otros congresistas de su partido.

Ejercicio 0:

En este ejercicio hicimos dos experimentos con para familiarizarnos con la herramienta Weka:

1) Correr nuestro dataset para ID3, J48 y RandomTree, por el momento con sus parámetros por defecto.

Empezamos usando el 100% de los datos para training y para el test hacemos Cross-validation, como las carpetas por defecto son 10 no hace falta usar el parámetro -x. Los comandos a la consola fueron:

```
java weka.classifiers.trees.J48 -t c:\dump\vote.arff -C 0.25 -M 2 >
c:\dump\ejercicio0\J48_100.txt

java weka.classifiers.trees.RandomTree -t c:\dump\vote.arff -K 1 -M
1.0 -S 1 > c:\dump\ejercicio0\RT_100.txt
```

Para usar solo el 33% de los datos para training y el 66% para test, usamos el parámetro split-percentage mediante el cual indicamos el porcentaje para training:

```
java weka.classifiers.trees.J48 -t c:\dump\vote.arff -split-percentage
33 -C 0.25 -M 2 > c:\dump\ejercicio0\J48_33.txt

java weka.classifiers.trees.RandomTree -t c:\dump\vote.arff -split-
percentage 33 -K 1 -M 1.0 -S 1 > c:\dump\ejercicio0\RT_33.txt
```

En los archivos de output (que están en la carpeta dump) el tamaño del árbol de decisión construido está expresado por el número de hojas y el número de nodos que tiene, etiquetados respectivamente como:

```
Number of Leaves: x
Size of the tree: x
```

Por otra parte, el desempeño del árbol de decisión construido, tanto con los datos de training como con de los de testing, está expresado por varios indicadores:

```
Correctly Classified Instances: x
Incorrectly Classified Instances: x
Kappa statistic: x
Mean absolute error: x
Root mean squared error: x
Relative absolute error: x
Root relative squared error: x
Total Number of Instances: x
```

Además, se presenta la matriz de confusión donde se detalla, para cada posible valor que puede tomar la clase, cuantas instancias con ese valor han sido clasificadas como tales y cuantas no (y con que otro valor se las clasificó).

2) Correr nuestro dataset para IBK, Naive Bayes, MultiLayerPerceptron, por el momento con sus parámetros por defecto.

Al igual que el experimento anterior, empezamos usando el 100% de los datos para training y para el test hacemos Cross-validation con 10 carpetas:

```
java weka.classifiers.lazy.IBk -t c:\dump\vote.arff -K 1 -W 0 -A
"weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\"" >
c:\dump\ejercicio0\IBK_100.txt

java weka.classifiers.bayes.NaiveBayes -t c:\dump\vote.arff >
c:\dump\ejercicio0\NaiveBayes_100.txt

java weka.classifiers.functions.MultilayerPerceptron -t
c:\dump\vote.arff -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a >
c:\dump\ejercicio0\MultilayerPerceptron_100.txt
```

Luego usamos el 33% de los datos para training y 66% para el test:

```
java weka.classifiers.lazy.IBk -t c:\dump\vote.arff -split-
percentage 33 -K 1 -W 0 -A
"weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\"" >
c:\dump\ejercicio0\IBK_33.txt

java weka.classifiers.bayes.NaiveBayes -t c:\dump\vote.arff -
split-percentage 33 > c:\dump\ejercicio0\NaiveBayes_33.txt

java weka.classifiers.functions.MultilayerPerceptron -t
c:\dump\vote.arff -split-percentage 33 -L 0.3 -M 0.2 -N 500 -V 0
-S 0 -E 20 -H a > c:\dump\ejercicio0\MultilayerPerceptron_33.txt
```

La performance del modelo está expresada por los mismos indicadores que en el experimento anterior.

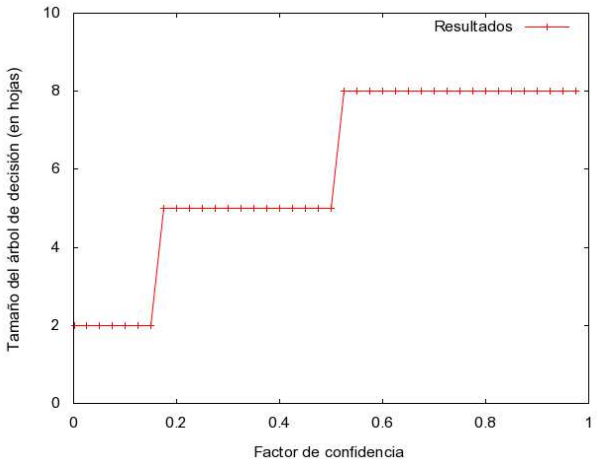
Ejercicio 1:

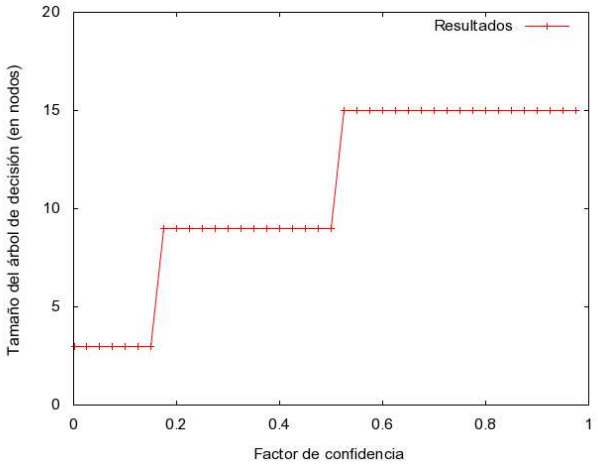
Este ejercicio tiene como objetivo ver el comportamiento de distintas técnicas de clasificación ante un sobreajuste. Realizamos diversos experimentos, siempre usando el 80% de los datos para training y el 20% para el test, cada uno con una técnica distinta.

En este ejercicio y en todos los demás para cada experimento haremos una breve descripción y luego iremos mostrando resultados obtenidos al tiempo que los analizamos.

1) Para J48 (o C4.5) variamos el factor de confianza de 0 (en realidad, un valor cercano a 0) a 1 en intervalos de 0.025.

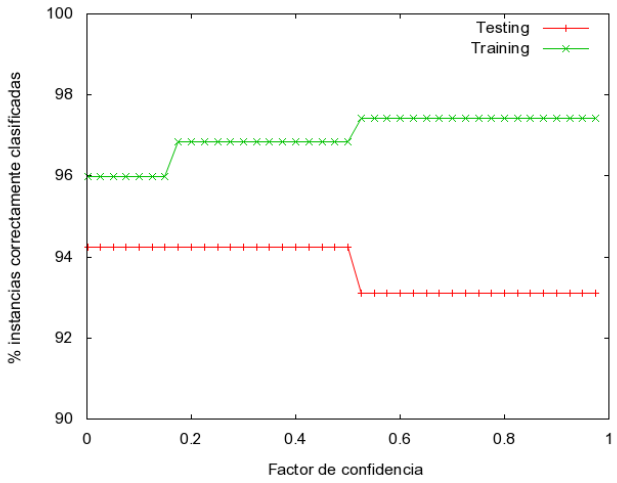
Este factor está relacionado con la confianza que desearíamos tener, ante una clasificación de una instancia, de que esta fue correcta en base a los datos de training. Es de esperarse que, cuanto mas grande sea el factor de confianza, menor sea la poda para el árbol de decisión resultante (y viceversa) ya que tratará de generalizar menos en vistas de equivocarse menos con los datos de training. De hecho, teniendo en cuenta que la poda reduce el tamaño del árbol:





Dado que existe una relación lineal entre el número de hojas de un árbol y el número de nodos, nos alcanzará con analizar solo uno de estos factores. Por eso, de ahora en más nos detendremos únicamente en el número de nodos.

Veamos ahora que ocurre con el porcentaje de instancias correctamente clasificadas, tanto para instancias de training como para las de testing, a medida que variamos el factor de confianza:

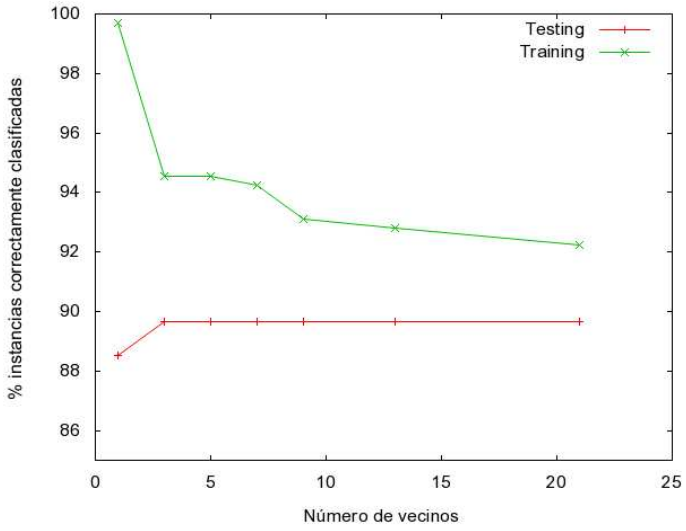


Como se mencionó al principio, a medida que el factor de confianza crece el modelo cada vez se vuelve menos general, esto es, se vuelve más acorde a los datos de training. Hemos visto también que en el árbol de decisión habrá más nodos (y hojas) con el objetivo de especializar la clasificación para cada una de las instancias de los datos de training. Así, el porcentaje de instancias correctamente clasificadas para los datos de training irá en aumento mientras que para los datos de testing, superado cierto umbral, comienza a disminuir.

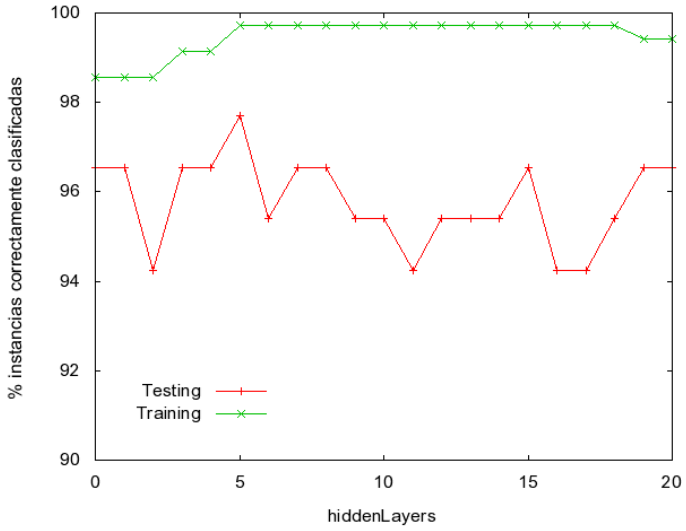
Al estar nuestro modelo sobreajustado (o sobreentrenado), no responderá adecuadamente ante posibles instancias del dominio nunca antes vistas que pueden diferir en cierto grado de las usadas para el training. Esto vale para este experimento pero también para los dos siguientes.

2) Para IBk fuimos variando el k, el número de vecinos, entre 1,3,5,7,9,13 y 21, siempre un número impar para que no se produzcan empates al momento de decidir que clase asignarle a una instancia basándonos en sus vecinos.

Al momento de hacer nuestro modelo, si solo nos basamos en un número pequeño de instancias para clasificar otra, dependeremos mucho de cuales son esas instancias. Luego, esperaremos que haya sobreajuste cuando el número de vecinos es pequeño. El siguiente gráfico refleja eso mismo:



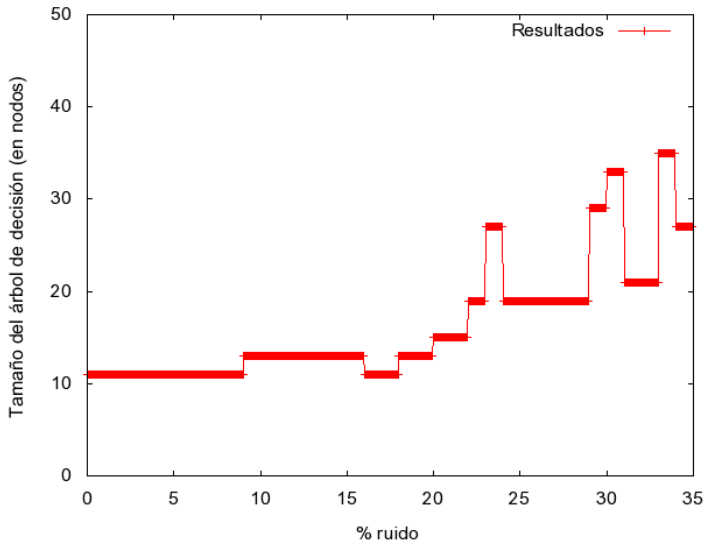
3) Para un perceptrón con múltiples capas hemos variado la cantidad de capas. Se puede observar en el gráfico que está a continuación que conforme aumenta la cantidad de capas el porcentaje de instancias clasificadas correctamente para los datos de training también aumenta mientras que para los datos de testing oscila. A diferencia de los experimentos anteriores, no encontramos fundamentos teóricos de redes neuronales que avalen este comportamiento.

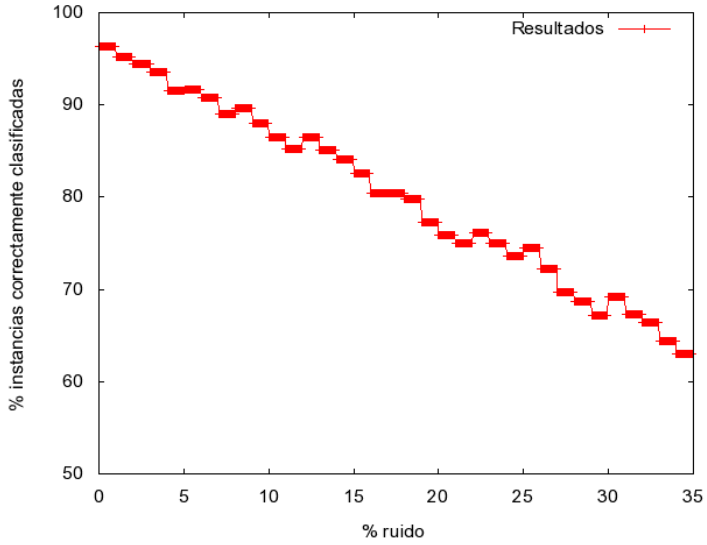


Ejercicio 3:

Este ejercicio tiene como objetivo ver el comportamiento de distintas técnicas de clasificación ante ruido en el dataset. Para eso, creamos datasets con un porcentaje de ruido que va desde 0 a 35 con intervalos de 0.05. Para todos los experimentos se uso la totalidad de los dataset creados para training y Cross-Validation con 10 carpetas para comprobar la efectividad de los distintos modelos.

1) Probamos todos los datasets con J48 viendo que ocurría con el tamaño del árbol de decisión y el porcentaje de instancias correctamente clasificadas.



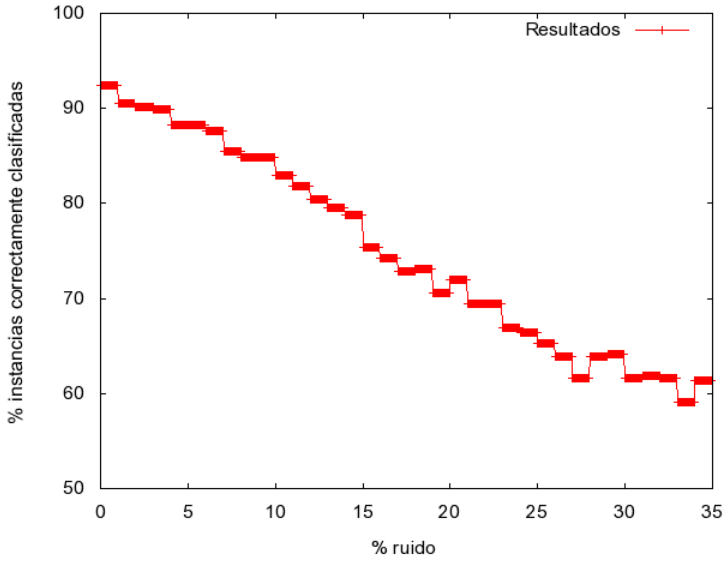


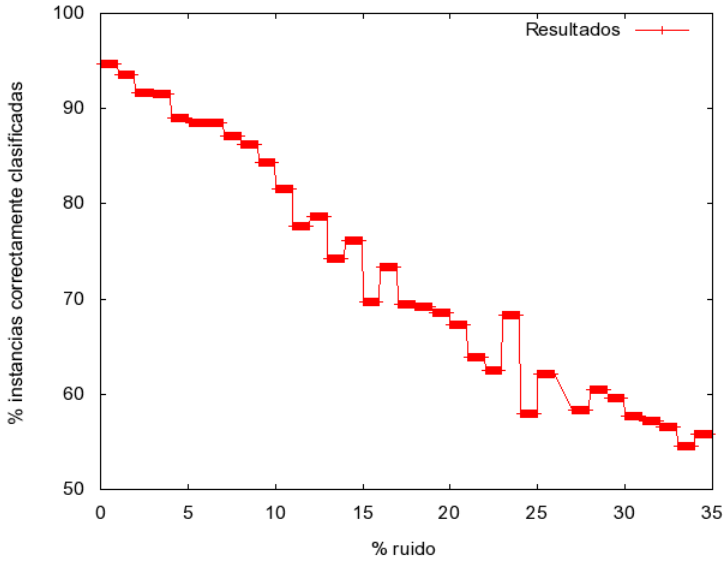
Por un lado, vemos que, a medida que el porcentaje de ruido aumenta, el árbol de decisión aumenta de tamaño. Esto se explica a partir de que el ruido introduce mayor diversidad de instancias y para clasificarlas con cierto factor de confianza, que permaneció fijo para todos los experimentos, hacen falta árboles cada vez más grandes.

Por otro lado, vemos que el porcentaje de instancias correctamente clasificadas se ve altamente afectado por el ruido. Una de las hipótesis en que se basan los algoritmos de clasificación, en particular J48, es que las instancias de un dataset están relacionadas estadísticamente. Al introducir ruido esta relación cada vez se difumina más y es de esperar que la efectividad de los modelos se reduzca significativamente.

Cabe destacar que tanto los datos de training como los datos para test estaban con ruido, de haber estado uno “limpio” y el otro “sucio” los resultados hubieran sido aún peores.

2) y 3) Los dos experimentos fueron como 1) pero para IBk y un perceptrón con múltiples capas. Veamos, en ese orden, los resultados:





Los parámetros para el perceptrón con múltiples capas fueron siempre los mismos que los del ejercicio 0. En cambio, para IBk se probó con k = 1, k = 5 y k = 10 ya que, según la bibliografía, más vecinos deberían proporcionar más tolerancia al ruido. Sin embargo, para nuestro problema en particular las diferencias en los resultados entre un k y otro fueron insignificantes.

El análisis que se puede hacer para estos resultados es el mismo que en 1) porque aunque los algoritmos sean distintos creemos que es la característica intrínseca a todos ellos mencionada en 1) la que lleva a tal degradación en la efectividad de los modelos ante el ruido.

Ejercicio 5:

El objetivo de este ejercicio, es poder identificar que mails son de spam y cuales son reales de la lista de alumnos/docentes de la facultad.

Para esto, se utilizan los diferentes algoritmos de clasificación provistos por el entorno Weka. Para poder llevar a cabo esto, fue necesario definir atributos, cuales creemos que pueden caracterizar y diferenciar a los mails de spam, de los mails que no son de spam.
Es importante marcar, que esta clasificación fue hecha para la lista de alumnos de la facultad, lo cual, limita el dominio del problema. Gracias esto pudimos definir algunos atributos, que sean caracteristicos de mails válidos/ inválidos dentro del entorno de la facultad, sin embargo, es posible que el comportamiento no sea el mismo en un dominio más amplio o distinto, como ser, cuenta de mails personales.

Implementación:
Para llevar a cabo nuestro objetivo, fue necesario generar un DataSet con los atributos que deseamos. Para esto, decidimos generar un parser de archivos eml.
Para parsear los archivos eml de este ejercicio, se decidió hacer un parser custom. El mismo recibe por parámetros atributos a testear, y en cada línea del mail, ejecuta un método del atributo, pasándole por parámetro el valor anterior. De esta forma, el atributo, decide el nuevo valor, teniendo en cuenta la nueva línea del mail, y el valor asignado en la iteración previa.

Modelo:
Fueron implementados 4 tipos de atributos:
Atributo básico: Dada la línea actual del mail, y el valor anterior definido para el mail, retorna el nuevo valor. El mismo usa la técnica de verificar si un string, es sub-string de la línea del mail.
Atributo acumulativo: Cuenta cantidad de apariciones de un cierto string. Luego el valor final, es definido teniendo en cuenta el intervalo del mismo.
Atributo Negador: Chekea que un string, no se encuentre en el mail.
Atributo Empty Line: Cuenta las líneas vacias.

Los atributos, tienen a su vez, un atributo, que les permite verificar si están buscando en la sección correspondiente.
Esto nos sirvió para poder tener una clase Generadora de DataSets abstraída de los diferentes tipos de atributos que puedan existir.

Algunos atributos comenzaron siendo positivos en la búsqueda, pero luego de ir descubriendo más atributos que mejoraban la clasificación, estos fueron perdiendo efecto, y terminaron siendo eliminados, como por ejemplo:

- Contar enters
- Contar cantidad de seteos de Font
- verificar si tiene attachement
- buscar atributo USER_IN_WHITELIST
- seteos de color
- otros atributos de html.

Atributos que ayudaron a mejorar la performance:

- Atributos para caracterizar mails de ventas, conteniendo palabras como: “venta”, “oferta”, “precio increíble”, etc.
- Atributos del tipo spam: “venus”, “familia”, “salud”, etc

- Atributos relacionados con mails de la facultad: “asamblea”, “tesis”, “concurso”, etc.
- Atributos que busque un mail de @hotmail.
- Atributo que contiene “considerado SPAM”, haciendo referencia a una ley de spam.
- Atributos que busquen .gif, .jpg.
- Atributos que busquen javascript, behavior.
- Subject general
- Subject Spam (palabras que pueden estar en un mail spam)
- Subject no Spam (palabras que pueden estar en un mail de la facultad)
- Contar tables en un mail
- Palabras válidas : aula, materia, etc
- HTMLValues : text/css , size=
- Mails con imágenes
- Mails con saludos

A continuación mostramos el porcentaje de clasificaciones correctas retornados en cada corrida.
El primer registro de la tabla, muestra el “Correctly Classified Instances” del DataSet con todos los atributos que decidimos utilizar.
Luego ejecutamos algoritmos con el weka, para la selección de atributos. El mismo retorna un subconjunto de atributos, y su respectivo ranquin. Para cada caso, mostramos los resultados obtenidos con los algoritmos IBK y J48.

	Correctly Classified Instances IBK	**Correctly Classified Instances J48**
DataSet todos los atributos	88.7615 %	83.945 %
Selección atributos CfsSubsetEval 1,5,6,8,9,10,12,13,14,15	84.633 %	82.7982 %
Selección atributos InfoGainAttributeEval 13,1,14,12,6,10,5,9,8,3,2,7,15	88.7615 %	84.633 %
Selección atributos ConsistencySubsetEval 1,2,3,5,6,7,8,9,10,12,13,14,15	88.7615 %	83.945 %
Selección atributos PrincipalComponents 1,2,3,4,5,6,7,8,9,10,15	79.5872 %	78.211 %
Selección atributos propia 6,7,8,9,12,13,14,15	84.1743 %	81.8807 %

Como conclusión, podríamos ver que la selección de atributos ConsistencySubsetEval, retorno un subconjunto de los atributos, y su performance fue la misma que el conjunto completo de atributos.

Aclaraciones:

- En los gráficos del ejercicio 1 llamamos, incorrectamente, factor de confidencia al factor de confianza.

- El ejercicio 2 sí fue hecho (ver dump\ejercicio2) aunque no se analizaron los resultados en este informe por falta de tiempo y falta de espacio en el mismo.

Bibliografía:

“Data mining, Practical Machine Learning Tools and Techniques with Java Implementations”; I. Witten, E. Frank; Morgan Kaufmann; 2000.