PRÁCTICA 5: Clasificación: Bayes Net, Multilayer Perceptron, SVM

Índice

1. Objetivos

Trabajar las siguientes competencias:

- **Competencias transversales:**
  - Trabajo autónomo
  - Trabajo en equipo: el equipo estará formado por 4 miembros
  - Pensamiento crítico
- **Competencias específicas:**
  - Capacidad de formular algoritmos de aprendizaje automático:
    - *Zero Rules* (`ZeroR`)
    - *One Rule* (`OneR`)
    - *k-Nearest Neighbors* (`IBk`)
    - *Support Vector Machines* (`SVM`)
    - *Multilayer Perceptron* (`MP`)
    - *Naive Bayes* (`NaiveBayesUdateable`)
    - *Bayes Network* (`BayesNet`)
    - Otros
  - Capacidad de implementar barridos de parámetros para estimar la mejor configuración de un algoritmo para una tarea.
  - En relación a la estimación de la calidad de un modelo predictor, capacidad para distinguir entre estimación realista y optimista.

2. Material

- Aplicación Weka
- Recursos accesibles desde Moodle:
  - Recursos generales: manual de la aplicación
  - Recursos específicos para la práctica: en Moodle, entre otros, hay disponibles los siguientes ficheros de datos:
    - `train.arff`
    - `dev.arff`
    - `test.arff`: se desconoce la clase de este conjunto, de hecho, éste es el conjunto de datos para el cual se quiere predecir la clase.

3. Marco teórico

3.1. *Baseline*

Antes de empezar con el trabajo grupal, cada integrante del grupo habrá explorado la tarea y los datos (número de instancias, número de atributos, tipo de atributos). Así mismo, cada integrante habrá determinado un umbral inferior de la calidad mínima que se le puede exigir al modelo que desarrollará el grupo. Es decir, cada integrante conocerá el *baseline* para la tarea.

El *baseline* es una cota pesimista (umbral inferior) de lo que se espera obtener en la tarea. En este caso podemos obtener un *baseline* empleando un modelo sencillo y que requiera poco tiempo de cómputo, por ejemplo, el modelo *One Rule* o el modelo *Naive Bayes*. El *baseline* sirve para comparar resultados (y por tanto, ayudan a elegir modelos): no tiene sentido utilizar un algoritmo complejo si no ofrece mejoras significativas respecto de uno más sencillo (*baseline*).

3.1.1. Marco teórico: algoritmo One Rule

El algoritmo *Zero Rules* [Witten et al., 2011, Sec. 11.4] (`ZeroR`), como se mencionó en la práctica 1, es un algoritmo muy elemental para hacer predicciones. La clase estimada para cualquier instancia de test (independientemente de sus atributos) es siempre la misma: la clase que tenga la máxima probabilidad a priori en el conjunto de entrenamiento (o el valor medio si la clase es numérica). En los problemas donde las instancias no están balanceadas entre todas las clases, este algoritmo ofrece una tasa real de aciertos o *accuracy* alta y frecuentemente se utiliza como *baseline* en estas circunstancias. No se puede justificar proponer otro algoritmo más complejo a menos que supere este *baseline*.

El algoritmo *One Rule*[1] [Witten et al., 2011, Sec. 4.1] (`OneR`) sí toma en cuenta los atributos, en realidad, sólo uno. Para estimar la clase de una instancia de test sólo toma en cuenta el valor de un atributo y predice la clase mediante una regla de decisión construida en base al valor de ese atributo. La regla de decisión consiste en mapear cada valor del dominio del atributo con uno de los valores de la clase. Cuando el atributo no toma valores discretos, emplea una estrategia para partir el dominio en sub-intervalos y mapear cada intervalo con uno de los valores de la clase. A modo de ejemplo, en la Figura 1 se muestran unos resultados obtenidos con este algoritmo para una tarea con 400 instancias caracterizada por dos atributos (*spectrum*, *flux*) y la clase (*classReaction*). El modelo *One Rule* ha seleccionado el atributo *flux* y ha propuesto la regla mostrada en la figura, con la cual se estima correctamente 319 de las 400 clases.

El algoritmo *One Rule* explora iterativamente cada uno de los atributos ($X_i \quad 1 \leq i \leq n$). Para cada valor del atributo que está explorando ($x \in Image(z_i^{(j)}) 1 \leq j \leq N = |\mathcal{Z}_{train}|$) genera una regla que asigna a ese valor del atributo la clase más votada entre las instancias con ese valor en ese atributo. Se elige el conjunto de reglas (asociadas a un atributo) que presente el menor ratio de error (ver detalles del algoritmo en el Apéndice A).

El algoritmo de aprendizaje *One Rule* consiste en determinar cuál de todos los atributos resulta más informativo para predecir la clase, y como sub-producto, cuáles son los intervalos que mejor ayudan a discriminar la clase en base al valor de ese atributo. Para ello calcula el error predictivo de cada atributo y selecciona aquel atributo con error predictivo

[1] Ver: `http://www.saedsayad.com/oner.htm`

```
Attributes:   3
              spectrum
              flux
              classReaction
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

flux:
        < 0.41579958692716384   -> Yes
        < 0.5147331207595894    -> No
        < 0.5673379374678738    -> Yes
        >= 0.5673379374678738   -> No
(319/400 instances correct)
```

```
if              flux <0.41  then   classReaction = Yes
elsif   0.41 <= flux <0.51  then   classReaction = No
elsif   0.51 <= flux <0.56  then   classReaction = Yes
elsif   0.56 >= flux        then   classReaction = No
end_if
return classReaction
```

Figura 1: Ejemplo: resultados de Weka con el modelo OneR e interpretación de las reglas.

mínimo. En definitiva, el algoritmo *One Rule* genera un árbol de decisión elemental con tan sólo un nivel de profundidad [Witten et al., 2011, Sec. 4.1].

Para calcular el error predictivo de cada atributo necesita operar sobre los posibles valores de un atributo, es decir, necesita que los atributos empleados sean discretos. Cuando los atributos no son discretos, el algoritmo cuenta con un modo para discretizarlos: crea intervalos garantizando que en cada intervalo habrá un número de instancias de la misma clase. Para ello se emplea un parámetro, que en Weka se designa: `B` (*the minimum bucket size for discretization*).

3.1.2. Marco teórico: algoritmo Naive Bayes

Ejercicio 1 ***algoritmo Naive Bayes***

1. *¿Cuál es el marco teórico del algoritmo Naive Bayes? [Alpaydin, 2010, Sec. 16.3.1]*
2. *¿Cuál es la diferencia entre las implementaciones de Weka del algoritmo Naive Bayes? [Witten et al., 2011, Sec. 11.4]*
   - *`NaiveBayesSimple`*
   - *`NaiveBayesUpdateable`*
   - *`NaiveBayesMultinomial`*
   - *`NaiveBayesMultinomial-Updateable`*
3. *Escribe el algoritmo de inferencia en pseudo-código.*

3.2. Otro algoritmo: MP, SVM, BN, etc.

Cada equipo de trabajo estudiará y sintetizará uno de los siguientes algoritmos, más sofisticados que los anteriores (los *baseline*):

- *Multilayer Perceptron* [Alpaydin, 2010, Chap. 11]
- *Support Vector Machines* [Alpaydin, 2010, Chap. 13]
- *Bayes Network* [Alpaydin, 2010, Chap. 16]
- Otros

Se recomienda consultar: [Alpaydin, 2010] y [Witten et al., 2011, Chap. 6].

4. Marco experimental

4.1. Diseño de software

Antes de empezar a implementar, se pide el diseño del proyecto, análisis de clases con la documentación detallada de cada clase y la distribución del trabajo por cada integrante del grupo.

4.2. Análisis de datos

Se dispone del siguiente conjunto de datos: (disponibles en Moodle)

- `train.arff`
- `dev.arff`
- `test.arff`: se desconoce la clase de este conjunto, de hecho, éste es el conjunto de datos para el cual se quiere predecir la clase.

Ejercicio 2 ***análisis de datos***

1. *Antes de empezar, explorar las características de los datos (presentados en la sección 4.2):*

```
java -cp /ClassPathTo/weka.jar weka.core.Instances train.arff
```

2. *Con los resultados obtenidos en el item 1, rellenar la tabla 1.*

	Raw data		
	Train	**Dev**	**Test**
Nominal Atributes			
Numeric Atributes			
Atributes total			
Instances $\oplus$			
Instances $\ominus$			
Instances total			

Tabla 1: Quantitative description of the original data set.

4.3. Pre-procesamiento

Aplicar los filtros que sean necesarios parar pre-procesar el conjunto de datos. Se pueden explorar, entre otros:

- Randomize: `weka.filters.unsupervised.instance.Randomize`
- Discretize: `weka.filters.unsupervised.attribute.Discretize`
- Normalize: `weka.filters.unsupervised.attribute.Normalize`
- Eliminar *outliers*. Quizá te resulten de utilidad los siguientes filtros:
  - `weka.filters.unsupervised.attribute.InterquartileRange` estableciendo el parámetro `outlierFactor = 1.5`
  - `weka.filters.unsupervised.instance.RemoveWithValues`
  - `weka.filters.unsupervised.attribute.Remove`
- Convertir todos los atributos de tipo nominal a binario: `weka.filters.unsupervised.attribute.NominalToBinary`
- Eliminar los atributos que varían poco o varían demasiado mediante `weka.filters.unsupervised.attribute.RemoveUseless` estableciendo el parámetro `maximumVariancePercentageAllowed` a `95`.
- Eliminar un porcentaje de instancias: `weka.filters.unsupervised.instance.RemovePercentage`

Además, en todos los casos se pide hacer una selección de atributos empleando alguna de las siguientes técnicas: `weka.filters.supervised.attribute.AttributeSelection`

1. *Attribute Evaluator:*
   - `weka.attributeSelection.CfsSubsetEval`
   - `weka.attributeSelection.ClassifierSubsetEval` estableciendo el clasificador que se utilice finalmente.
2. *Search Method:*
   - `weka.attributeSelection.GreedyStepwise`
   - `weka.attributeSelection.LinearForwardSelection`

Ejercicio 3 *Una vez terminado el pre-proceso (sección 4.3), se pide rellenar la tabla 2*

	Pre-processed data		
	Train	**Dev**	**Test**
Nominal Atributes			
Numeric Atributes			
Atributes total			
Instances $\oplus$			
Instances $\ominus$			
Instances total			

Tabla 2: Quantitative description of the pre-processed data set.

4.4. Inferencia del modelo y estimación de la calidad esperada

Cada equipo de trabajo estudiará y sintetizará uno de los siguientes algoritmos:

- *Multilayer Perceptron* [Alpaydin, 2010, Chap. 11]
- *Support Vector Machines* [Alpaydin, 2010, Chap. 13]
- *Bayes Network* [Alpaydin, 2010, Chap. 16]
- Otros

Se recomienda consultar: [Alpaydin, 2010] y [Witten et al., 2011, Chap. 6].

4.5. Barrido de parámetros

Se obtendrán los parámetros óptimos del modelo según un esquema de tipo *hold-out*: con el conjunto `train.arff` para entrenar el modelo y el conjunto `dev.arff` para evaluarlo. Se elegirán los parámetros que optimicen la *f-measure* de la clase minoritaria. El barrido se hará sobre los 2 parámetros más representativos para el modelo. Como resultado de este apartado se obtienen los "parámetros óptimos", es decir, aquellos parámetros que hacen que el modelo ofrezca los mejores resultados según el criterio establecido.

Ejercicio 4 *En la sección 4.5 se lleva a cabo un barrido de parámetros mediante un esquema de evaluación* hold-out *(con los conjuntos* `train.arff` *y* `dev.arff`*) y como criterio de optimización la* f-measure *de la clase minoritaria.*

1. *Explicar en qué rango se ha hecho le barrido de parámetros y cuál ha sido el salto. Así mismo, indicar qué valor toman los parámetros óptimos.*
2. *¿Cuáles son las figuras de mérito que se obtienen con este esquema hold-out (train vs. dev)? Completa la tabla 3*

		Hold out		
	Class	**Precision**	**Recall**	**F-Measure**
Baseline	$\oplus$			
	$\ominus$			
	W.Avg.			
Model	$\oplus$			
	$\ominus$			
	W.Avg.			

Tabla 3: Performance Performance of each of the two models explored, namely, Baseline and Model on a hold-out evaluation scheme with train vs. dev.

4.6. Inferencia del modelo final

Establece los parámetros óptimos y entrena un modelo con el conjunto de datos supervisados completo: `train_dev.arff` donde `train_dev.arff` $\leftarrow$ {`train.arff`} $\bigcup$ {`dev.arff`}.

Esta rutina ofrecerá como sub-producto la calidad estimada del modelo según los esquemas de evaluación:

1. Evaluación no-honesta
2. *10-fold cross validation*

La calidad se medirá mediante las figuras de mérito detalladas por clase y su media ponderada (*Detailed Accuracy By Class*).

Así mismo, se pide guardar el modelo resultante en un fichero binario: eg. `NaiveBayes.model`. Las siguientes líneas de código muestran cómo guardar el modelo (para más información consultar: `https://weka.wikispaces.com/Serialization`):

```
// Serialize model (save model)
SerializationHelper.write(modelPath, estimador);
```

Ejercicio 5 *En la sección 4.6 se ha entrenado un modelo con el conjunto de datos supervisados completo.*

1. *Explicar en qué rango se ha hecho le barrido de parámetros y cuál ha sido el salto. Así mismo, indicar qué valor toman los parámetros óptimos.*
2. *¿Cuáles son las figuras de mérito que se obtienen con este esquema? Completa la tabla 4 con tantos modelos como hayas implementado.*

		Resubstitution error			10-fold Cross Validation		
	Class	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Baseline	$\oplus$						
	$\ominus$						
	W.Avg.						
Model	$\oplus$						
	$\ominus$						
	W.Avg.						

Tabla 4: Performance of each of the two models explored, namely, Baseline and Model on two evaluation schemes: re-substitution error (this provides the upper threshold achievable) and 10-fold cross validation.

4.7. Clasificación

Finalmente, cargar el modelo binario para hacer las predicciones del conjunto de test, las siguientes líneas de código muestran cómo guardar el modelo (para más información consultar: `https://weka.wikispaces.com/Serialization`):

```
// Deserialize model (load model)
estimador = (NaiveBayes) SerializationHelper.read(modelPath);
```

Como resultado se ofrecerá la instancia, junto con la clase estimada en un fichero de salida: `test.predictions.txt`.

5. Entregables

En las siguientes secciones se resumen los resultados a entregar y se indican, explícitamente, el nombre y los formatos. Se recomienda utilizar la herramienta LyX para editar el informe. En Moodle hay plantillas disponibles en LaTeX y en LyX (hay gran variedad de plantillas en `http://www.ctan.org`).

5.1. Software

- Diseño: equema del diseño del software: `design.pdf`
- Proyecto: `workspace.tgz`: el código fuente del proyecto comprimido para dos algoritmos (baseline y otro más sofisticado) [2]. Huelga decir que se espera un programa modular, sin dependencias respecto de los datos, que reciba los parámetros por línea de comando y que sea auto-explicativo (si no recibe los argumentos esperados lanzará una excepción informando sobre el objetivo del y los argumentos que requiere). El paquete de software incluirá las siguientes funcionalidades especificadas en la sección 4.

[2] Compresión en `tar.gzip` o `bz2`, no se admiten `.rar`

- Ejecutables:
  1. Preproceso: `Preprocess.jar`
  2. Inferencia: `GetModel.jar`
  3. Clasificación: `Classify.jar`
- `Readme.txt`: el paquete de software entregado estará debidamente documentado, de modo que sea fácil de utilizar para cualquier usuario (por ejemplo una guía para compilar y ejecutar el software). Se indicarán explícitamente las pre-condiciones y post-condiciones. En principio, el ejecutable no debería tener dependencias a ficheros del usuario y debería ser independiente del sistema operativo, en cualquier caso, si tiene alguna dependencia debería indicarse explícitamente. Suele ser de utilidad incluir un ejemplo de uso: dado un conjunto de datos detallar el procedimiento para obtener los resultados experimentales mencionados en el apartado anterior.

Alternativamente, si se emplea `weka.classifiers.meta.FilteredClassifier`, se pueden ofrecer un solo ejecutable que aúne el pre-proceso y la inferencia del modelo. Para más detalles, consultar: `https://weka.wikispaces.com/Use+Weka+in+your+Java+code`

5.2. Resultados experimentales

En un paquete (`ExperimentalResults.tgz`) se incluirán los resultados experimentales obtenidos con el software diseñado. Se incluirá la evaluación o estimación de la calidad de cada modelo y la estimación de la clase para el conjunto de test con cada modelo:

- **Evaluación:** Se pide estimar la calidad de cada uno de los dos algoritmos considerados en los siguientes ficheros respectivamente:
  `EvaluationBaseline.txt` y `EvaluationAlgorithm.txt`.
  Estos ficheros incluirán:
  - Estimación mediante *hold-out*, según se indica en la sección 4.5.
  - Cota realista de la calidad esperada mediante *10-fold Cross Validation*, según se indica en la sección 4.6.
  - Cota superior de la calidad esperada mediante el método de evaluación no-honesto (según se indica en la sección 4.6).
- **Predicciones:** Así mismo, se pide estimar la clase del conjunto de test con cada uno de los dos algoritmos considerados:
  - Baseline: se entregará un fichero (`TestPredictionsOneR.txt`) con la clase estimada por el *baseline* para cada instancia del conjunto de test.
  - Algoritmo: se debe entregar un fichero (`TestPredictionsAlgorithm.txt`) con la clase estimada por el algoritmo para cada instancia del conjunto de test.

5.3. Documentación

Se pide entregar un informe de prácticas en formato pdf (`informe.pdf`) que incluya los contenidos enumerados a continuación:

1. **Portada:** Título (algoritmo abordado). Miembros del equipo.
2. **Marco teórico:** (aprox. 4 pgs.)

   a) Documentación: trabajo de documentación y síntesis sobre el algoritmo abordado (indicando explícitamente las referencias bibliográficas en las que se apoya).
      - Baseline *Naive Bayes* (ver ejercicio 1)
      - *Support Vector Machines* (`SVM`)
      - *Multilayer Perceptron* (`MP`)
      - *Bayes Network* (`BayesNet`)
   b) Diseño: así mismo, se presentará el diseño del software que se ha hecho para abordar este problema. También se pueden dar los detalles de implementación que se consideren relevantes (ejercicios 5, 4).

3. **Marco experimental:** resumir el trabajo de diseño y resultados experimentales obtenidos (aprox. 8 pgs.)
   a) Datos: descripción cualitativa y cuantitativa del conjunto de datos y ofrecer en este punto los resultados del ejercicio 2.
   b) Pre-proceso elegido: explicar la motivación subyacente a cada filtro aplicado. En este punto ofrecer los resultados del ejercicio 3.
   c) Resultados experimentales: evaluación de la calidad que puede ofrecer el modelo. Resumir los resultados que se espera obtener con el *baseline* y con el modelo estudiado según marco de evaluación *5-fold cross validation*. En base a las figuras de mérito obtenidas, discutir cuál de los dos modelos presenta un mejor rendimiento (baseline vs. estudiado). Todos los resultados se ofrecerán tabulados, cada tabla llevará un pie con un índice que será referenciado a lo largo del texto. Es muy importante razonar las respuestas, analizar los resultados, discutir si son consistentes etc.
   d) Ejemplo de ejecución de los ejecutables (descritos en la sección 5.1).
4. **Conclusiones y trabajo futuro:** resumir las fortalezas y las debilidades del proyecto y dar nociones indicaciones sobre cómo podría mejorarse. (aprox. 1 pg.)
5. **Bibliografía:** la bibliografía se referencia para reforzar el contenido y apoyarlo en trabajos de otros autores. Es crucial citar las fuentes en el punto en el que se recurre a ellas, por ejemplo, siempre que las afirmaciones no sean del autor del trabajo o estén basadas en otros trabajos. No se puede olvidar hacer una referencia a la fuente de las figuras siempre que no sean originales. Un autor no debería limitarse a escribir una sección con una lista de fuentes bibliográficas sobre el tema. Por contra, esta sección sólo incluirá bibliografía a la que se haya recurrido para hacer el trabajo y que haya sido citada en algún punto del trabajo para dejar claro el contexto en el que se recurre a ella.
6. **Valoración subjetiva**: (voluntaria) en un apéndice puedes añadir una reflexión sobre la tarea realizada, los siguientes puntos pueden servir de guía (no es necesario incluirlos todos, alternativamente, se pueden incluir otros):
   a) ¿Ha alcanzado el equipo los objetivos que se plantean?
   b) ¿Cuánto tiempo se ha trabajado en esta tarea en promedio? Desglosado: estudio del algoritmo y búsqueda bibliográfica, diseño e implementación de software, informe,...
   c) ¿Ha resultado de utilidad para el equipo la tarea planteada?
   d) ¿Qué aspectos de la tarea han despertado mayor interés? Sugerencias para mejorar la tarea: sugerencias para que se consiga despertar mayor interés y motivación.

5.4. Plazos de entrega

Los plazos de entrega vienen detallados en Moodle.

Referencias

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. MIT Press.

[Witten et al., 2011] Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems, 3rd edition.

Apéndices

A. Algoritmo *One Rule*

```
algorithm OneRule
require   Z_train , x
          Z_train : set of classified instances
          x and the query instance to be classified
          Let N = |Z_train| = |{z^(1), ..., z^(j), ..., z^(N)}| = |{z^(j)}_{j=1}^N|
          Let z^(j) = (z_1^(j), ... z_i^(j), ... z_n^(j), c^(j)) ∈ X_1 × ··· × X_i × ··· × X_n × C
          Let C = {C_1, ..., C_m}
ensure    ĉ(x_q) ∈ C
begin
          // 1. Generate rules for each attribute
          for(i=1; i ≤ n; i++ ):
              // find all different values
              V ← {z_i^(j)}_{j=1}^N = {v_1, ..., v_d}
              // for each value of that attribute make a rule as follows:
              for(j=1; j ≤ d; j++ ):
                  rule[i][j] ← arg máx_{C_k ∈ C} N_{C_k} ( 1 ≤ r ≤ N ∧ z_i^(r) = v_j ∧ c^(r) = C_k )
              done
              //calculate the error rate of the rules on the attribute
              ER[i] ← ErrorRate( rule[i][j] , Z_train)
          done
          // 2. Chose the rules (attribute) with the smallest error rate
          s ← arg mín_{i:1≤i≤n} ER[i]
          // 3. Apply rule
          r ← j :   1 ≤ j ≤ d ∧ x_s = v_j
          ĉ(x_q) ← rule[s][r]
end
```

A modo de ejercicio individual para adquirir destreza en la formulación de algoritmos de aprendizaje automático, se recomienda[3] desglosar el algoritmo en dos: uno para entrenar el modelo y otro para hacer las predicciones. Expresar los algoritmos en pseudo-código y dar una breve explicación sobre los métodos que intervienen.

[3] Esta parte no se calificará, no forma parte del lote de entregables de esta práctica.

B. Técnicas de discretización

El algoritmo presentado en la sección A está formulado para trabajar con atributos que tomen valores discretos. No obstante los atributos de tipo continuo (en Weka se refieren como numéricos) se pueden convertir en discretos (en Weka se refieren como nominales) recurriendo a técnicas de discretización [Witten et al., 2011, Sec. 4.1]. Respecto de las técnicas de discretización, la taxonomía distingue dos grandes familias:

- **Discretización no-supervisada:**
  - *Equal width binning:* se puede discretizar una variable numérica haciendo particiones de igual anchura en el intervalo y asignando a la partición la clase mayoritaria de la misma. Esto puede dar lugar a intervalos no etiquetados (aquellos donde no hay instancias). Frecuentemente, da lugar a empate en zonas con numerosas instancias.
  - *Equal frequency binning:* para evitar el sobre-ajuste se pueden establecer los subintervalos de forma que en todos ellos haya el mismo número de muestras. Nuevamente, a cada sub-intervalo se le asigna la clase más votada localmente (moda en el sub-intervalo).
- **Discretización supervisada:**
  - *Trigger binning:* Otra forma de hacer las particiones en el rango de un atributo numérico es añadir fronteras cada vez que cambie el valor de la clase para ese atributo. Esto evita tener particiones vacías. Sin embargo esta técnica suele generar un número elevado de particiones y en ocasiones una sola muestra rodeada de muchas de otra clase es suficiente para generar un nuevo intervalo.
  - *Trigger binning with minimum frequency threshold:* siempre que instancias de dos sub-intervalos contiguos tengan asociados la misma clase se pueden asimilar al otro intervalo (así se gana precisión en la definición de las fronteras). De este modo, para hacer las particiones se impone un límite inferior en el número de muestras que tiene que haber de una clase (la clase mayoritaria) para permitir formar una partición.

Ejercicio 6 *1. ¿Por qué se llama discretización "supervisada" y "no-supervisada"?*

2. *Consulta los ejemplos de discretización no-supervisada:*
   - `http://www.saedsayad.com/unsupervised_binning.htm`
3. *Consulta los ejemplos de discretización supervisada:*
   - `http://www.saedsayad.com/supervised_binning.htm`
4. *Practica con Weka las dos familias de técnicas de discretización. Antes de empezar, consulta:*
   - `http://weka.wikispaces.com/Discretizing+datasets`
5. *¿Guardan relación las técnicas de discretización que ofrece Weka y las presentadas en* `http://www.saedsayad.com` *?*