

PRÁCTICA 4: Text Mining: Sentiment Analysis, SPAM classification

Índice	
1. Objetivos	2
2. Material	2
3. Marco teórico	3
3.1. Text Mining	3
3.2. <i>Baseline</i>	3
3.3. Algoritmo propuesto: SVM, Bagging, Tree, BN	4
4. Marco experimental	5
4.1. Descripción de la tarea	5
4.2. Pre-procesamiento	5
4.2.1. Formato del conjunto de datos: arff	5
4.2.2. Espacio de atributos: Bag of Words	6
4.2.3. Selección de atributos mediante técnicas de ganancia de información	9
4.2.4. Selección de atributos mediante técnicas TF-IDF	10
4.3. Inferencia del modelo y Ajuste de parámetros	12
4.3.1. Barrido de parámetros	12
4.3.2. Inferencia del modelo final	13
4.4. Clasificación	14
5. Entregables	15
5.1. Software	15
5.2. Resultados experimentales	15
5.3. Documentación	16
5.4. Plazos de entrega	17

1. Objetivos

Trabajar las siguientes competencias:

■ Competencias transversales:

- Trabajo en equipo: el equipo estará formado por 3-4 miembros máximo.
- Pensamiento crítico.

■ Competencias específicas:

- Capacidad de solventar un problema real de *text mining*.
- Capacidad de entender y utilizar algoritmos de aprendizaje automático:
 - Baseline: *Naïve Bayes* (bayes → NaïveBayes)
 - Algoritmos propuestos:
 - ◊ *Bagging* (meta → Bagging)
 - ◊ *RandomForest*
 - ◊ *Support Vector Machines* (functions → LibSVM o SMO)
 - ◊ *Bayes Network* (BayesNet)
- Capacidad de aplicar las distintas técnicas de pre-procesamiento de datos.
- Capacidad de implementar barridos de parámetros para estimar la mejor configuración de un algoritmo para una tarea.
- Adquirir habilidad en la utilización de librerías de minería de datos pre-diseñadas.
- Capacidad de diseñar un estudio experimental, ejecutarlo, y analizar los resultados así como extraer conclusiones.

2. Material

- Aplicación Weka: <http://www.cs.waikato.ac.nz/ml/weka>
- Wiki de la aplicación: <http://weka.wikispaces.com/Text+categorization+with+Weka>
- Recursos accesibles desde eGela:
 - Recursos generales: manual de la aplicación.
 - Recursos específicos para la práctica:
 - Ficheros de datos para la práctica: *spam*, *filmReviews*, *tweetSentiment*
 - ◊ *readme*: contiene la descripción de los datos.
 - ◊ *train*: conjunto de datos supervisado en bruto (no está en formato *arff*)
 - ◊ *test*: conjunto de datos no-supervisado. Se desconoce la clase de las instancias, y es precisamente lo que se quiere predecir. Este conjunto también está en bruto.
- Recursos bibliográficos sobre *text mining* (*spam classification*, *sentiment analysis*, *polarity*, *etc.*): *W. Richert, L. P. Coelho, Building Machine Learning Systems with Python. PACKT. July 2013.* [Richert and Coelho, 2013].

3. Marco teórico

En esta sección se proporciona el fundamento teórico de la práctica.

3.1. Text Mining

El *Text mining* es un campo de la minería de datos en la cual los conjuntos de datos provienen textos (paginas web, foros, correos electrónicos, opiniones, noticias, etc.). Es una disciplina que está ligada a dos áreas de conocimiento muy significativos en ciencias de la computación: el procesamiento de lenguaje natural y la inteligencia artificial. El objetivo del *text mining* consiste en que la máquina sea capaz de extraer conocimiento a partir de textos. En la actualidad, el *text mining* es una disciplina al alza, y empresas como Google, IBM-Watson o diarios de noticias como *The Wall Street Journal* están apostando fuerte por ellas.

En esta práctica se propone una tarea de *text mining*. Dados diferentes conjuntos de datos (provenientes de diferentes dominios) se pedirá predecir si un mail es SPAM o no; dada la sinopsis de una película predecir su género; o predecir si un mensaje de twitter, un *tweet*, es **positivo**, **neutral** o **negativo**.

Ejercicio 1 *Ejercicios introductorios a la temática:*

1. Definir la tarea: antes de comenzar con la práctica, analizar el conjunto de datos y recabar y resumir toda la información relativa.
2. Encuentra relaciones entre Text mining y las empresas: (*KDnugets, BI, DSS resources, AI Topics*)
 - Encuentra empresas que desarrollen productos que impliquen text mining.
 - Encuentra empresas que desarrollen librerías y softwares para text mining.
 - Encuentra empresas que utilicen productos de text mining.
3. Trabajar con textos implica retos que no existen cuando tratamos con conjuntos de datos numéricos. Propón un ejemplo que ilustre este hecho [Richert and Coelho, 2013].

3.2. 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 *Naïve 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*).

Ejercicio 2 *El algoritmo Naïve Bayes*

1. Describe formalmente el algoritmo Naïve Bayes [Alpaydin, 2010, Sec. 16.3.1].
2. ¿Cual es la diferencia entre las diferentes implementaciones que Weka contiene del Naïve Bayes? [Witten and Frank, 2011, Sec. 11.4]
 - *NaïveBayesSimple*
 - *NaïveBayesUpdateable*
 - *NaïveBayesMultinomial*
 - *NaïveBayesMultinomial-Updateable*
3. Dado un conjunto de datos supervisado, describe el método de inferencia (puedes utilizar un pseudocódigo).
4. Dado el modelo predictor, describe el procedimiento de clasificación de una nueva instancia (puedes utilizar un pseudocódigo).

3.3. Algoritmo propuesto: SVM, Bagging, Tree, BN

Además del *baseline*, se propondrá una algoritmo más avanzado.

Ejercicio 3 *El algoritmo propuesto* Cada equipo de trabajo estudiará y sintetizará uno de los siguientes algoritmos, más sofisticado que el *baseline*:

- [Support Vector Machine](#) (*LibSVM* edo *SMD*) [Alpaydin, 2010, Chap. 13]
- [Bagging](#) [Alpaydin, 2010, Chap. 17]
- [Random Forest](#) (*RandomForest*)
- [Bayes Network](#) (*BayesNet*) [Alpaydin, 2010, Chap. 16]

Se recomienda consultar: [Alpaydin, 2010] y [Witten and Frank, 2011, Chap. 6].

4. Marco experimental

En esta sección se detallará los apartados que el software tendrá que implementar: pre-procesamiento de los datos, inferencia del modelo óptimo, aplicación del modelo predictor...

4.1. Descripción de la tarea

Antes de empezar, es importante recopilar toda la información que tengamos sobre la tarea. En ocasiones, los datos son confidenciales y el programador no puede saber más de lo que se puede extraer de un análisis de datos como es el caso.

Ejercicio 4 *Analiza el conjunto de datos y responde a las siguientes preguntas:*

1. *Analiza el readme que viene junto a los datos, y describe la tarea.*
2. *Revisa el conjunto de datos y trata de extraer características cuantitativas: el número de textos, la longitud de cada texto, el número de textos que aparecen repetidos.*
3. *Anota todo elemento característico o reseñable que habrá que tener en cuenta en posteriores procedimientos.*

4.2. Pre-procesamiento

El pre-procesamiento de los datos implicará realizar las siguientes tareas:

- Convertir el conjunto de datos en bruto al formato arff (sección 4.2.1).
- Representar los mensajes en el espacio BOW mediante la utilización de atributos numéricos (sección 4.2.2).
- Selección de atributos: tendremos disponibles dos técnicas:
 - Selección de los atributos más relevantes mediante el cálculo de la información mutua (sección 4.2.3).
 - Selección de atributos mediante TF-IDF (sección 4.2.4).

4.2.1. Formato del conjunto de datos: arff

Los datos se han descargado de páginas web, y por lo tanto decimos que está en bruto (*raw*). El primer paso del pre-procesado consiste en adecuarlos a la herramienta con los que vamos a trabajar. En este caso, diseñaremos e implementaremos un software (`getARFF.jar`) que convierta el conjunto de datos al formato arff. Dado el fichero `train` (disponible en eGela), obtendremos el fichero `train.arff`. En el caso de que no introduzcamos ningún argumento, se imprimirá por pantalla las indicaciones de su funcionamiento.

```
java -jar getARFF.jar train train.arff
```

Después de aplicar esta transformación, obtendremos tres ficheros de datos:

- train.arff
- dev.arff
- test.arff

Ejercicio 5 *Análisis del conjunto de datos*

1. Describe los datos en base al fichero readme.
2. Explorar las características de los datos con el siguiente y otros comandos:

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

3. Con los resultados obtenidos en el ítem 2, rellenar la tabla 1.

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

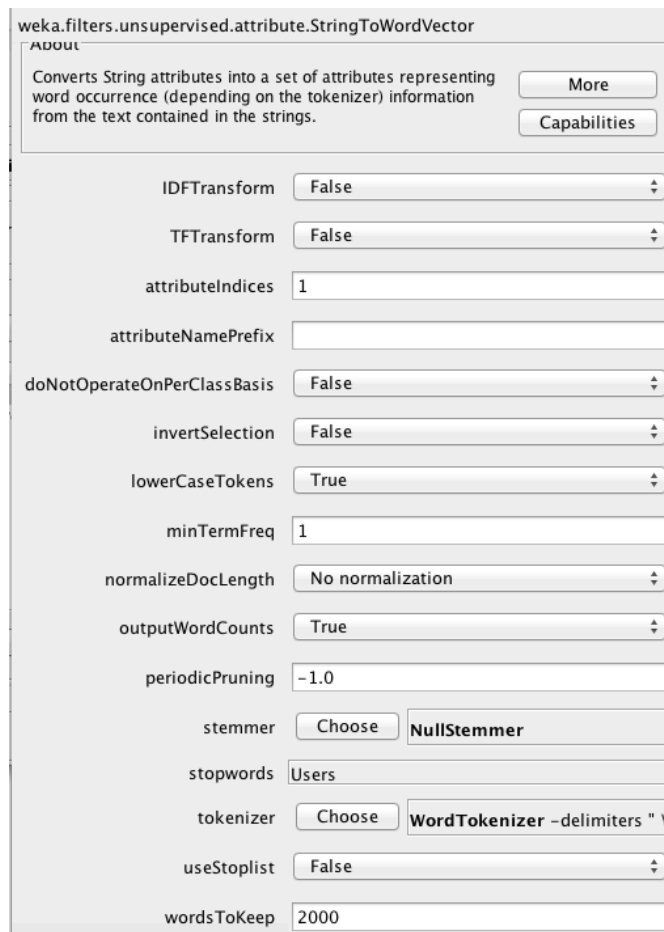
Tabla 1: Quantitative description of the original data set.

4.2.2. Espacio de atributos: Bag of Words

Cada instancia representa un texto (mail, tweet...) y lo representa mediante un único atributo (denominado **text**) que está asociado a un atributo clase (denominado **class**). En este punto se desea convertir el atributo del mensaje en un vector numérico, para ello se aplicará el filtro `weka.filters.unsupervised.attribute.StringToWordVector`. Este filtro convierte un atributo de tipo **String** en un conjunto de atributos que representan la presencia de las palabras en el texto. Concretamente, la dimensión del vector será el número de palabras presentes en la tarea (el vocabulario de la aplicación) o alternativamente se puede restringir mediante el parámetro `wordsToKeep`. El contenido del vector está determinado por el parámetro `outputWordCounts`:

- **outputWordCounts=True**: en este caso el contenido de la posición j en la instancia i es un valor entero que indica el número de veces que aparece la palabra j en el mensaje i
- **outputWordCounts=False**: en este caso el contenido de la posición j en la instancia i será un valor booleano (0 ó 1) que indica si la palabra j está presente o ausente en el mensaje i
 - 1 si la palabra está presente en el mensaje
 - 0 si la palabra no está presente en el mensaje

También puede resultar de interés considerar idénticas las palabras escritas en mayúsculas o en minúsculas estableciendo la opción: **lowerCaseTokens=True**. En este apartado tanto **TFTransform** como **IDFTransform** se establecen a **False**. En la Figura 1, se muestra de forma gráfica el valor de los parámetros mencionados (sin embargo, en la práctica se establecerán en el propio programa, no gráficamente).



weka.filters.unsupervised.attribute.StringToWordVector

ABOUT

Converts String attributes into a set of attributes representing word occurrence (depending on the tokenizer) information from the text contained in the strings.

More

Capabilities

IDFTransform False

TFTransform False

attributeIndices 1

attributeNamePrefix

doNotOperateOnPerClassBasis False

invertSelection False

lowerCaseTokens True

minTermFreq 1

normalizeDocLength No normalization

outputWordCounts True

periodicPruning -1.0

stemmer Choose NullStemmer

stopwords Users

tokenizer Choose WordTokenizer -delimiters " \

useStoplist False

wordsToKeep 2000

Figura 1: Transformar atributos de tipo String en *bag of words*

Una vez aplicado el filtro, cada instancia se caracteriza por un número elevado de atributos numéricos, aproximadamente tantos como palabras se hayan permitido (mediante el parámetro

`wordsToKeep`). El identificador del atributo es la palabra y el valor del atributo (numérico o booleano dependiendo del parámetro `outputWordCounts`) indica la presencia de la palabra en la instancia.

Una vez aplicado el filtro sobre un conjunto de datos, el resultado es una matriz dispersa¹ (se puede convertir en no-dispersa mediante el filtro `weka.filters.unsupervised.instance.SparseToNonSparse`).

En este segundo apartado del pre-procesamiento de los datos, se pide diseñar e implementar un software (`arff2bow.jar`), que dado el conjunto `train.arff` transforma las instancias a la representación BOW y las guarda en el fichero `trainBOW.arff`.

```
java -jar arff2bow.jar train.arff trainBOW.arff
```

Ejercicio 6 Una vez transformados los datos al espacio BOW, rellena la siguiente tabla.

	Pre-processed data: BOW		
	Train	Dev	Test
Nominal Attributes			
Numeric Attributes			
String Attributes			
Boolean Attributes			
Attributes total			
Instances \oplus			
Instances \ominus			
Instances total			

Tabla 2: Quantitative description of the data set represented in terms of BOW.

Nota: Esta técnica para representar textos se conoce como *Bag of Words* (BOW). Sólo toma en cuenta la presencia/ausencia de las palabras pero no toma en cuenta el orden de esas palabras en el texto. En este punto ya no se dispone del texto con una estructura sintáctica. No se conoce el orden en el que aparecen las palabras en el *tweet*, sólo qué palabras aparecen. Por ejemplo la cadena “el pez grande se come al pequeño” tiene una representación BOW idéntica a “el pez pequeño se come al grande”. Con esta transformación se ha perdido parte de la información pero se ha conseguido una representación de las instancias más simple y manejable computacionalmente.

¹*Sparse ARFF*: <http://weka.wikispaces.com/ARFF>

4.2.3. Selección de atributos mediante técnicas de ganancia de información

Cada instancia se representa mediante un conjunto de atributos numéricos (aproximadamente tantos como **wordsToKeep**). Cada atributo representa la presencia o ausencia de términos clave en el mensaje. Es decir, la información disponible para caracterizar la instancia es sólo si un término (una palabra) está presente o no en un documento (texto, sms, e-mail).

En las tareas de minería datos en texto, el hecho de emplear las palabras como atributos provoca que la dimensión del espacio de atributos sea muy elevada. Se dispone de un número muy elevado de atributos para representar una instancia y típicamente, el ratio de aparición de los atributos en el conjunto de instancias es muy bajo, dando lugar a dos fenómenos perjudiciales: sobre-ajuste y sesgos en las estadísticas.

Por este motivo, las técnicas de selección de atributos (*Feature Subset Selection* o FSS) se utilizan para descartar atributos redundantes o irrelevantes para el proceso de clasificación. Ayudan a simplificar la representación de las instancias reduciendo el orden de magnitud del espacio de búsqueda. Como ventaja, al reducir la dimensión del espacio de búsqueda, se suele agilizar el proceso de clasificación (tanto en entrenamiento como en test). Sin embargo esta simplificación también conlleva pérdida de información.

Feature subset selection se encuentra en Weka en el apartado **Select attributes** o en el apartado **Preprocess** → **Filter** → **AttributeSelection**.

1. **Attribute Evaluator** [Witten and Frank, 2011, Sec. 11.8] : En esta práctica, se seguirá el criterio de ganancia en información, i.e. **InfoGainAttributeEval**.
2. **Search Method**: se especifica la estrategia para buscar el subconjunto de atributos. En esta práctica establecerá el método de búsqueda por ranking (**Ranker**). Se pide establecer los parámetros que se consideren más oportunos para el **Ranker** argumentando la decisión. Los parámetros establecidos por defecto son:

```
Ranker -T -1.7976931348623157E308 -N -1
```

Donde:

- **numToSelect** (opción **-N**): número de atributos que se desea mantener donde **-1** indica que se pretenden mantener todos.
- **threshold** (opción **-T**): el umbral inferior del valor que tiene que presentar un atributo en la métrica adoptada para hacer la selección en **Attribute Evaluator** a fin de no ser descartado. Por defecto, se establece **-1.7976931348623157E308**, el mínimo Long integer en Java.

Para realizar este apartado se parte del fichero: **trainBOW.arff** y mediante las técnicas descritas, se eliminarán los atributos redundantes e irrelevantes. Concretamente, dado el fichero **trainBOW.arff**, se pide diseñar e implementar un software (**fssInfoGain.jar**) que permita reducir el espacio de atributos. Al finalizar este apartado se obtendrá el fichero: **trainBOW_FSS_InfoGain.arff**.

```
java -jar fssInfoGain.jar trainBOW.arff trainBOW_FSS_InfoGain.arff
```

Ejercicio 7 Analiza los datos **trainBOW_FSS_InfoGain.arff** y rellena la siguiente tabla.

Pre-processed data: BOW and FSS			
	Train	Dev	Test
Nominal Atributes			
Numeric Atributes			
String Atributes			
Bolean Atributes			
Atributes total			
Instances \oplus			
Instances \ominus			
Instances total			

Tabla 3: Quantitative description of the data set represented in terms of BOW after having selected the features according to Information Gain criteria.

4.2.4. Selección de atributos mediante técnicas TF-IDF

Para realizar este apartado se parte del fichero `trainBOW.arff` y se vuelve a aplicar el filtro `stringToWordVector` pero en este caso, con las opciones TF-IDF como se describirá después de dar el marco teórico de esta técnica. Aplicada la selección de atributos se obtendrá el fichero `trainBOW_FSS_TFIDF.arff`.

Se define la frecuencia relativa del término w_i en el documento d_j , *term frequency* (TF), según la expresión (1):

$$TF(w_i, d_j) = \frac{f(w_i, d_j)}{\sum_{w_i \in V} f(w_i, d_j)} \quad (1)$$

donde:

- $f(w_i, d_j)$: representa el número de veces que aparece el término w_i en el documento d_j
- $\sum_{w_i \in d} f(w_i, d_j)$: representa el número total de términos que aparecen en el documento d_j (representa el número total de palabras del documento, $|W|$, no el número de palabras distintas)
- V : el conjunto de términos o vocabulario de la aplicación, es decir, $V = \{w_1, \dots, w_i, \dots\} \subseteq W$ donde $w_i \neq w_j \forall i \neq j$ (conjunto de términos distintos en el conjunto de documentos)

Intuitivamente, cuanto mayor sea $TF(w_i, d_j)$, más característico o relevante resulta el término w_i para describir el documento d_j . Sin embargo, los términos frecuentes como determinantes o artículos son muy frecuentes en todos los documentos, y por tanto no son buenos atributos predictores. Para atenuar la relevancia que se le asocia al término w_i se define la frecuencia relativa de los documentos que contienen el término w_i , *document frequency* (DF), según la expresión (2):

$$DF(w_i) = \frac{\sum_{d_j \in D} \delta(w_i, d_j)}{|D|} \quad (2)$$

donde:

- $\delta(w_i, d_j)$: representa la delta de Kronecker sobre la pertenencia del término w_i en el documento d_j según indica la expresión (3).

$$\delta(w_i, d_j) = \begin{cases} 1 & w_i \text{ está presente en el documento } d_j \\ 0 & w_i \text{ no está presente en el documento } d_j \end{cases} \quad (3)$$

- $\sum_{d_j \in D} \delta(w_i, d_j)$: representa el número de documentos que contienen el término w_i
- $|D|$: representa el número total de documentos

Intuitivamente, cuanto menor sea $DF(w_i, d_j)$ (o de forma equivalente, cuanto mayor sea el inverso, es decir, cuanto mayor sea $DF(w_i, d_j)^{-1}$) más ayudará el término w_i a discriminar entre los distintos documentos. A fin de determinar cuantitativamente el grado de relevancia del término w_i en el conjunto de documentos se define TF-IDF (*term frequency - inverse document frequency*)² según la expresión (4):

$$TF - IDF(w_i, d_j) = TF(w_i, d_j) \cdot \log \frac{1}{DF(w_i)} \quad (4)$$

En resumen, TF es una medida para cuantificar la relevancia de un término dentro de un documento. IDF es una medida para cuantificar la relevancia de un término en un conjunto de documentos. TF-IDF es una medida que combina TF e IDF, de modo que cuantifica la relevancia de un término dentro de un documento considerando los demás documentos.

En definitiva, cuanto mayor sea la frecuencia del término w_i en el documento d_j y cuanto menor sea la frecuencia del término w_i en el conjunto de documentos mayor será $TFIDF(w_i, d_j)$. De este modo, para los determinantes, artículos y otros términos frecuentes en muchos documentos, el TFIDF toma un valor cercano a 0. TF-IDF es una métrica típicamente utilizada para cuantificar lo discriminante que resulta el término w_i .

Este proceso sirve para establecer qué términos utilizar en problemas de minería de texto. Dado que los términos son los atributos de cada instancia, TF-IDF es una técnica que sirve para seleccionar los atributos más relevantes para clasificar el conjunto de documentos. Para profundizar en estos conceptos se recomienda consultar las siguientes fuentes:

- *Stanford University Nature Language Processing Video Playlist 19: Ranked Information Retrieval* by D. Jurafsky and C. Manning [ver video](#) ³
- “Introduction to Information Retrieval”, C. Manning, P. Raghavan and H. Schütze, Cambridge University Press. 2008. [leer IR book](#) ⁴

En Weka se puede encontrar en el filtro `StringToWordVector` (ya se utilizó en la sección 4.2.2), pero en esta ocasión estableciendo `outputWordCounts=True`, `TFTransform` y `IDFTransform` a `True`. El resultado es una matriz dispersa (se puede convertir en no-dispersa mediante el filtro `weka.filters.unsupervised.instance.SparseToNonSparse`).

²TF-IDF: ¡no denota la resta de las dos magnitudes! de hecho, está relacionada con el producto

³<http://opencourseonline.com/273/stanford-university-nature-language-processing-video-playlist-19-ranked-information-retrieval>

⁴<http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html>

```
java -jar fssTFIDF.jar trainBOW.arff trainBOW_FSS_TFIDF.arff
```

Ejercicio 8 Analiza el fichero *trainBOW_FSS_TFIDF.arff* y rellena la siguiente tabla.

	Pre-processed data: BOW and TF-IDF		
	Train	Dev	Test
Nominal Atributes			
Numeric Atributes			
String Atributes			
Boolean Atributes			
Atributes total			
Instances \oplus			
Instances \ominus			
Instances total			

Tabla 4: Quantitative description of the data set represented in terms of BOW after having selected the features according to TF-IDF criteria.

4.3. Inferencia del modelo y Ajuste de parámetros

Cada grupo de trabajo, además del algoritmo *baseline*, trabajará con uno de los siguientes algoritmos de aprendizaje automático:

- *Support Vector Machine* (LibSVM edo SMO) [Alpaydin, 2010, Chap. 13]
- *Bagging* [Alpaydin, 2010, Chap. 17]
- *Random Forest* (RandomForest)
- *Bayes Network* (BayesNet) [Alpaydin, 2010, Chap. 16]

Se recomienda consultar: [Alpaydin, 2010] y [Witten and Frank, 2011, Chap. 6].

4.3.1. 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 9 En la sección 4.3.1 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 el 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 5

	Class	Hold out (train vs. dev)		
		Precision	Recall	F-Measure
Baseline	\oplus			
	\ominus			
	W.Avg.			
Model	\oplus			
	\ominus			
	W.Avg.			

Tabla 5: Performance Performance of each of the two models explored, namely, Baseline and Model on a hold-out evaluation scheme with train vs. dev.

4.3.2. Inferencia del modelo final

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

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

1. Evaluación no-honesta (*resubstitution error*).
2. *10-fold cross validation*
3. *Hold-out* (opcional). Para ello divide el conjunto supervisado completo, **train_dev.arff**, de forma aleatoria en las siguientes proporciones: %70 para entrenar %30 para evaluar. Filtros que te pueden resultar útiles:

- `weka.filters.unsupervised.instance.Randomize`
- `weka.filters.unsupervised.instance.RemovePercentage`

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 10 En la sección 4.3.2 se ha entrenado un modelo con el conjunto de datos supervisados completo. ¿Cuál es la calidad esperada para este modelo? Completa la tabla 6 con tantos modelos como hayas implementado. Añade el resultado obtenido mediante el esquema Hold-out si es que has implementado ese esquema de evaluación y responde a la siguiente pregunta: ¿coinciden estos resultados con los obtenidos en la sección 4.3.1? ¿por qué?

	Class	Resubstitution error			10-fold Cross Validation		
		Precision	Recall	F-Measure	Precision	Recall	F-Measure
Baseline	⊕						
	⊖						
	W.Avg.						
Model	⊕						
	⊖						
	W.Avg.						

Tabla 6: 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.

4.4. 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 eGela hay plantillas disponibles en LaTeX y en LyX (hay gran variedad de plantillas en <http://www.ctan.org>).

5.1. Software

- Diseño: esquema del diseño del software: `design.pdf`
- Proyecto: `workspace.tgz`: el código fuente del proyecto comprimido para dos algoritmos (baseline y otro más sofisticado) ⁵. 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.
- Ejecutables: Alternativamente, si se emplea `weka.classifiers.meta.FilteredClassifier` junto con `weka.filters.MultiFilter`, 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>
 1. Preproceso: `getARFF.jar`, `arff2bow.jar`, `getARFF.jar`, `fssInfoGain.jar`, `fssTFIDF.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.

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:

⁵Compresión en `tar.gz` o `bz2`, no se admiten `.rar`

- Estimación mediante *hold-out*, según se indica en la sección 4.3.1.
- Cota realista de la calidad esperada mediante *10-fold Cross Validation*, según se indica en la sección 4.3.2.
- Cota superior de la calidad esperada mediante el método de evaluación no-honesto (según se indica en la sección 4.3.2).
- **Predicciones:** estimar la clase del conjunto de test con cada uno de los dos algoritmos considerados y guardar el resultado en los ficheros:
 - Baseline: `TestPredictionsBaseline.txt`
 - Algoritmo: `TestPredictionsAlgorithm.txt`

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. Fecha.
2. **Marco teórico:**
 - a) Documentación: trabajo de documentación y síntesis sobre el algoritmo abordado (indicando explícitamente las referencias bibliográficas en las que se apoya). (aprox. 4 pgs.)
 - Baseline *Naive Bayes* (ver ejercicio 2)
 - *Support Vector Machine* (LibSVM edo SMO)
 - *Bagging*
 - *Random Forest* (RandomForest)
 - *Bayes Network* (BayesNet)
 - b) Diseño: así mismo, se presentará el diseño del software que se ha hecho para abordar este problema y la distribución de las tareas. También se pueden dar los detalles de implementación que se consideren relevantes (ejercicios 10, 9).
3. **Marco experimental:** (aprox. 8 pgs.)
 - a) Datos: descripción cualitativa y cuantitativa del conjunto de datos y ofrecer en este punto los resultados del ejercicio 5.
 - b) Pre-proceso elegido: explicar la motivación subyacente a cada filtro aplicado.
 - 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. 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 eGela.

Referencias

- [Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.
- [Orallo et al., 2004] Orallo, J. H., Ramirez, M., and Ferri, C. (2004). *Introducción a la Minería de Datos*. Pearson Educación.
- [Richert and Coelho, 2013] Richert, W. and Coelho, L. P. (2013). *Building Machine Learning Systems with Python*. PACKT.
- [Witten and Frank, 2011] Witten, I. H. and Frank, E. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems, 3rd Edition, San Francisco, CA, USA.