PROYECTO: Text Mining
Parte 1: Representación de datos

Índice

1. Material

Recursos accesibles desde eGela:

- Recursos generales: manual de la aplicación.
- 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.

2. Objetivos

- **Competencias transversales:**
  - Pensamiento crítico.
  - Trabajo en equipo
- **Competencias específicas:**
  - Capacidad de representar documentos mediante distintos tipos de caracterizaciones (e.g. raw, BoW, TF·IDF)
  - Capacidad de dar una descripción operativa de datos consistente para conjuntos de entrenamiento y evaluación
- **Competencias profesionales:** aplicación real de *business intelligence* en *text mining*, detección de spam.

3. Contexto de aplicación: Minería de Texto

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`, `neutro` o `negativo`.

Ejercicio 1 *Ejercicios introductorios a la temática:*

1. *Encuentra relaciones entre* Text mining *y las empresas: (fuentes de información útiles: KDnugets, BI, DSS resources, AI Topics)*
2. *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].*

Para profundizar en este contexto se sugieren los siguientes recursos:

- Minería de textos - marco teórico: *W. Richert, L. P. Coelho, Building Machine Learning Systems with Python. PACKT. July 2013.* [Richert and Coelho, 2013].
- Minería de textos con Weka: wikispaces `http://weka.wikispaces.com/Text+categorization+with+Weka`
- Descripción operativa de datos: [Witten et al., 2011]

4. Marco experimental: pre-proceso

4.1. Representación raw

Los datos pueden venir de fuentes diversas (e.g. web) y en formatos dispares, como es el caso de los conjuntos de datos disponibles para esta práctica. El primer paso en el pre-procesado consiste en adaptar los datos en bruto (*raw*) al formato requerido por la herramienta (Weka, en nuestro caso). En este punto, se pide implementar software (`getARFF.jar`) para hacer la conversión del conjunto de documentos a formato arff. La metodología empleada dependerá, estrechamente, de la fuente de datos. Se trata de una implementación *ad-hoc* para un conjunto de documentos concreto. Como resultado, se pide generar un fichero arff donde cada instancia representa un documento (mail, tweet, ...) y lo representa mediante un único atributo (denominado `text`) que está asociado a un atributo clase (denominado `class`).

Ejercicio 2 *Dado el conjunto `train` se pide generar el fichero `train.arff` (igualmente para los conjuntos `dev` y `test`).*

```
java -jar getRawARFF.jar train train.arff
```

Ejercicio 3 ***Análisis de datos:*** *se desea documentar los datos disponibles para la tarea, es decir, los obtenidos en el ejercicio 2:*

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

1. *Analiza el* readme *que viene junto a los datos, y describe la tarea.*
2. *Explora el siguiente comando para obtener información básica de los datos (instancias, atributos,* missing*,* unique*,* distinct *etc.) y con los resultados obtenidos completa la tabla 1.*

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

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

Tabla 1: Quantitative description of the original data set.

4.2. Transformación del espacio de atributos

4.2.1. Bag of Words: BoW

En el formato raw, cada instancia representa un documento (mail, tweet, ...) y lo representa mediante un único atributo (denominado `text`) de tipo `string`. En este punto se desea representar cada documento mediante un conjunto de atributos, preferentemente, numéricos. Para ello se aplicará el filtro `weka.filters.unsupervised.attribute.StringToWordVector` que convierte un atributo de tipo `string` en un vector de atributos (en la Figura 1, se muestra de forma gráfica, según la GUI). La dimensión del vector será el número de palabras presentes en la tarea (el vocabulario de la aplicación determinado por el conjunto de entrenamiento) 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

A fin de reducir la dimensión del vector de atributos puede resultar de interés considerar idénticas las palabras escritas en mayúsculas o en minúsculas. El filtro `StringToWordVector` lo implementa estableciendo la opción: `lowerCaseTokens=True`. En la Figura 1 se muestran, de forma gráfica, los parámetros mencionados.

Una vez aplicado el filtro, `StringToWordVector` cada documento viene representado mediante un vector "no-disperso" (consultar la sección "*Sparse ARFF*" en: `http://weka.wikispaces.com/ARFF`). No todos los clasificadores son capaces de emplear este tipo de representación *NonSparse*. Mediante el filtro `weka.filters.unsupervised.instance.NonSparseToSparse` se puede convertir en un vector disperso (*Sparse*).

Ejercicio 4 ***Sparse vs. Non-Sparse***

1. *Una vez aplicado el filtro* `StringToWordVector` *guarda el conjunto de datos resultante en un fichero* `dataBOW_NonSparse.arff`*. Edita el fichero y describe cómo se representan las instancias. ¿Vienen todas las instancias descritas con el mismo número de atributos?*
2. *Aplica el filtro* `NonSparseToSparse` *al conjunto anterior y guarda el conjunto de datos resultante en un fichero* `dataBOW_Sparse.arff`*. Nuevamente, edita el fichero y describe cómo se representan las instancias. ¿Vienen todas las instancias descritas con el mismo número de atributos?*
3. *¿Qué diferencia hay entre las representaciones Sparse y NonSparse?*

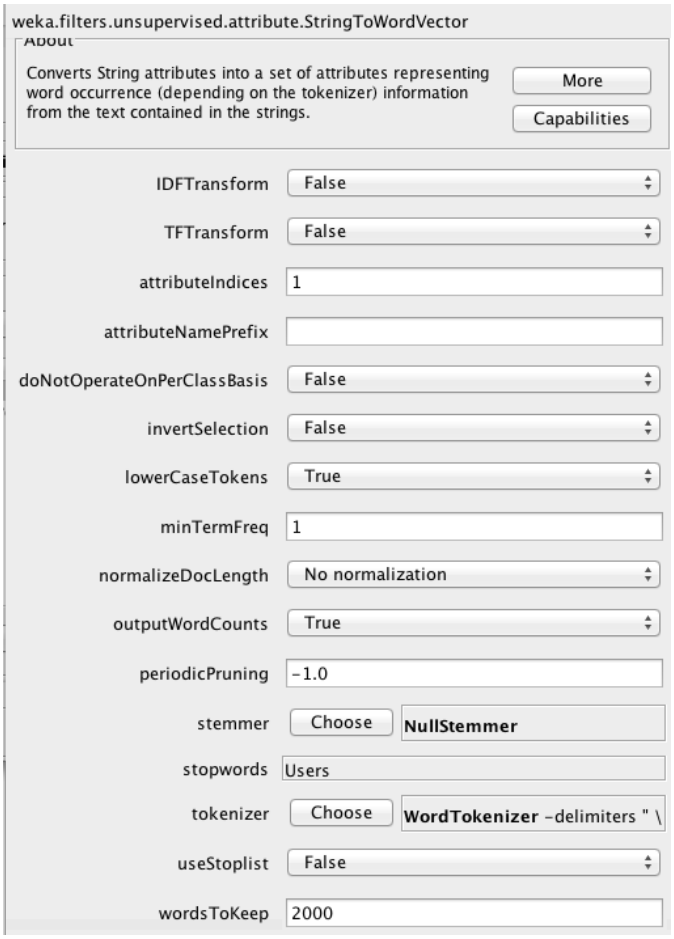

Figura 1: Transformar atributos de tipo String en *bag of words*

Ejercicio 5 ***Raw to BoW***

Implementa un programa que permita obtener la representación BoW Sparse a partir del conjunto raw. En este apartado se emplea primero el filtro `StringToWordVector`*, donde tanto* `TFTransform` *como* `IDFTransform` *se establecen a* `False`*, y después el filtro* `NonSparseToSparse`*.*

```
java -jar raw2BoW.jar   train.arff   trainBOW.arff
```

Observación: 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 documento, 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.

4.2.2. Term Frequency, Inverse Document Frequency: TF·IDF

La representación BoW tiende a emplear un número alto de atributos. La dimensión del vector viene establecida por el número de palabras presentes en el vocabulario del conjunto de entrenamiento. Además, no indica qué términos discriminan mejor la clase (e.g. spam vs. no-spam), quizá sería suficiente revisar, únicamente, la presencia de ciertos términos y reducir así la dimensión del espacio de atributos. TF·IDF ofrece una representación que da respuesta a los aspectos anteriores. A continuación describiremos el marco teórico TF·IDF, es importante para comprender la representación resultante.

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

$$TF(w_i, d_j) = \frac{f(w_i, d_j)}{\sum\limits_{w_i \in V} f(w_i, d_j)} \tag{1}$$

donde:

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

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 que son muy frecuentes en todos los documentos (e.g. determinantes o artículos) 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):

$$DF(w_i) = \frac{\sum\limits_{d_j \in \mathcal{D}} \delta(w_i, d_j)}{|\mathcal{D}|} \tag{2}$$

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).

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

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

Intuitivamente, cuanto menor sea $DF(w_i)$, más ayudará el término w_i a discriminar entre los distintos documentos. De forma equivalente, cuanto mayor sea el inverso, es decir, cuanto mayor sea $DF(w_i)^{-1}$ mejor discriminante será el término w_i.

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*) según la expresión (4):

$$TF \cdot IDF(w_i, d_j) = TF(w_i, d_j) \cdot \log \left[\frac{1}{DF(w_i)} \right] \tag{4}$$

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.

Ejercicio 6 ***Raw to TF·IDF***
En Weka se puede encontrar en el filtro `StringToWordVector` *(ya se utilizó en la sección 4.2.1), pero en esta ocasión estableciendo* `outputWordCounts=True`, `TFTransform` *y* `IDFTransform` *a* `True`. *Implementa un programa que permita obtener la representación TF·IDF NonSparse a partir del conjunto raw.*

```
java -jar raw2TFIDF.jar   train.arff   trainTFIDF.arff
```

Observación: Dado que TF·IDF nos ofrece una medida que indica qué términos son más relevantes en problemas de minería de texto, entonces, se puede emplear TF·IDF, de forma indirecta, para seleccionar el conjunto de atributos predictores (*Feature Subset Selection*). 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 ▷ [1]
- "Introduction to Information Retrieval", C. Manning, P. Raghavan and H. Schütze, Cambridge University Press. 2008. leer IR book ▷[2]

4.3. Adaptar la representación del test al espacio de entrenamiento

Ejercicio 7 *Estableciendo el espacio de atributos de la tarea:*

1. *Dado* `train.arff`, *genera* `trainBOW.arff`
2. *Con el mismo filtro genera* `devBOW1.arff` *y* `testBOW1.arff`
3. *Prueba a entrenar un clasificador (e.g. Naive Bayes) con el conjunto* `trainBOW.arff`, *trata de estimar la calidad empleando el conjunto de evaluación* `devBOW1.arff`. *¿Hay problemas de incompatibilidad?*

[1] http://opencourseonline.com/273/stanford-university-nature-language-processing-video-playlist-19-ranked-information-retrieval
[2] http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.htm

4. *Antes de seguir adelante comprueba si son compatibles los espacios de atributos de los ficheros* `train.arff` *y* `devBOW1.arff`*. ¿Hay el mismo número de atributos? ¿Están los atributos en el mismo orden en ambos conjuntos?*

Como sabemos, las instancias del conjunto de entrenamiento vienen descritas mediante un espacio de atributos. Cuando se infiere un modelo predictor, el modelo será capaz de hacer predicciones sólo en ese espacio de atributos. Típicamente, como resultado del ejercicio 7 se presenta un problema de incompatibilidad dado que disponemos de cada conjunto (train, dev, test) representado en un espacio distinto. En la fase de pre-proceso de datos hay que adaptar los datos de dev y test al conjunto de datos de entrenamiento, en este caso, `trainBOW.arff`. Por lo tanto, necesitamos crear conjuntos `devBOW.arff` y `testBOW.arff` compatibles con `trainBOW.arff`.

Ejercicio 8 *Eel fichero* `trainBOW.arff` *determina el espacio de atributos (número de atributos, identificador, tipo, orden etc.). Se pide implementar software para representar un ficheros raw de evaluación, e.g.* `dev.arff`*, en BOW compatible con el* `trainBOW.arff` *pero sin alterar el formato del conjunto de entrenamiento. El software podría recibir tres argumentos, 1. el conjunto* `trainBOW.arff` *con el que se establece el espacio de atributos; 2. el path del fichero de entrada que se desea transformar; 3. el path de salida donde almacenar el fichero transformado. Diseña tests para verificar el correcto funcionamiento del software.*

```
java -jar getBOWCompatible.jar trainBOW.arff dev.arff devBOW.arff
```

Ejercicio 9 *Una vez transformados los conjuntos train, dev y test al espacio BOW, completa la tabla 2.*

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

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

5. Resultados

Se pide **entregar** un paquete de software (`Preprocess.tgz`) y un informe preliminar (`informePreprocess.pdf`) que se detallan a continuación.

5.1. Software

Como resultado de la fase de pre-proceso de datos se piden **paquetes** software independientes que sirvan, respectivamente, para abordar los siguientes objetivos:

1. `GetRaw` Convertir una colección de documentos a formato arff raw. Se trata de software ad-hoc. Dada una colección de documentos obtendrá un fichero arff donde cada instancia hará referencia a un documento y vendrá representada con un atributo (denominado text) de tipo `string` y la clase (denominada class) de tipo nominal (e.g. spam, NOspam).
2. `TransformRaw` Transformar el espacio de atributos del conjunto de entrenamiento a BoW o TF·IDF (según se indique en los argumentos) permitiendo dar como salida una representación Sparse o NonSparse (indicado mediante argumentos).
3. `MakeCompatible` Dado un conjunto de evaluación (e.g. `dev.arff`) representarlo en un espacio de atributos compatible con el de entrenamiento (sea BoW o TF·IDF tanto Sparse como NonSparse).

De cada paquete se **entrega**:

1. El código fuente (e.g. cada proyecto comprimido)
2. Un ejecutable (e.g. `GetRaw.jar`, `TransformRaw.jar`, `MakeCompatible.jar`)

Especificaciones del software:

- Todos los paquetes de software estarán documentados (JavaDoc).
- Al ejecutarlos sin argumentos mostrarán el modo de funcionamiento, es decir, ofrecerán al usuario una breve documentación:
  1. objetivo del programa
  2. pre-condición
  3. post-condición
  4. lista de argumentos con su descripción
  5. ejemplo de uso en la línea de comandos

5.2. Informe preliminar

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

1. **Portada:** Miembros del equipo. Fecha.
2. **Sección: Introducción a la minería de textos** (aprox. 1 pgs.)
   - ¿En qué consiste la minería de textos?
   - Contextualizar la minería de textos en el ámbito de inteligencia artificial.

- Ejemplos donde se aplica
- Retos que entraña la minería de textos frente a otro tipo de datos

3. **Sección: Pre-procesamiento**(aprox. 2 pgs.)
   - Resumir las características más relevantes de los tipos de datos raw, BoW o TF·IDF
   - Ejemplos de cada transformación. Para un documento dado, mostrar su representación raw, BoW y ·IDF.
   - Detalles relevantes del diseño y la implementación del software, así como la distribución de tareas.
4. **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.

El **formato** es libre, sin embargo, se anima a utilizar la herramienta LyX o editores LaTeX para editar el informe. En eGela hay plantillas disponibles en LaTeX y en LyX (hay gran variedad de plantillas en `http://www.ctan.org`) y también hay plataformas on-line que disponen de plantillas y no requieren instalar software (e.g. `http://www.overleaf.com`).

5.3. Plazos de entrega

Los plazos de entrega vienen detallados en eGela.

Referencias

[Richert and Coelho, 2013] Richert, W. and Coelho, L. P. (2013). *Building Machine Learning Systems with Python*. PACKT.

[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.