

2. PROYECTO: Clustering de documentos

| | |
|---|-----------|
| Índice | |
| 1. Introducción | 2 |
| 2. Objetivos | 3 |
| 3. Material | 3 |
| 4. Metodología de trabajo | 3 |
| 5. Pre-proceso | 4 |
| 5.1. Formato | 4 |
| 5.2. Espacio de decisión | 6 |
| 5.2.1. Bag of Words | 6 |
| 5.2.2. TF-IDF | 8 |
| 5.2.3. <i>Word Embeddings</i> | 8 |
| 5.3. Filtrado | 11 |
| 6. Clustering | 12 |
| 6.1. <i>Hierarchical clustering</i> | 12 |
| 6.2. <i>k-means clustering</i> | 13 |
| 6.3. Índices de calidad | 14 |
| 6.4. Aplicación del modelo inferido | 14 |
| 7. Casos prácticos de aplicación | 14 |
| 7.1. Clustering de posts | 15 |
| 7.2. Reuters | 15 |
| 7.3. Clustering semántico | 15 |
| 7.4. Otras | 16 |
| 8. Resultados | 17 |

| | |
|---|-----------|
| 8.1. Entregables | 17 |
| 8.2. Evaluación | 18 |
| 8.3. Plazos | 18 |
| A. Imágenes tomadas vía satélite | 20 |
| A.1. Descripción de la tarea | 20 |
| A.2. Procedimiento | 21 |
| A.2.1. Clustering | 21 |
| A.2.2. Representación PCA | 22 |
| B. Marco teórico: Bag of Words | 24 |
| B.1. Dataset: Amazon | 25 |
| C. Marco teórico: TF-IDF | 26 |
| C.1. Python: TF-IDF | 27 |

1. Introducción

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* (*spam classification, sentiment analysis, polarity, etab.*), concretamente buscamos agrupar documentos (e.g. noticias, tweets, mails, etc) similares. Es decir, clustering the documentos. Las disponemos de grandes colecciones de documentos, texto escrito (e.g. wikipedia) y queremos descubrir agrupaciones de documentos relacionados, de este modo, para un documento dado, podemos ofrecer documentos similares. Si un documento de texto es una instancia de mi problema ¿cómo la caracterizamos? ¿qué atributos podemos emplear para describir un documento?

La caracterización de texto es el primer reto que presenta la minería de textos y en este proyecto se proponen diversas alternativas. También se estudian diversos algoritmos de clustering. En técnicas exploratorias, como el clustering, es crucial la visualización de resultados. Como objetivo en este contexto, se debe primar una presentación de resultados de los algoritmos de clustering así como una evaluación intuitiva, preferentemente de forma gráfica.

2. Objetivos

Trabajar las siguientes competencias:

■ Competencias transversales:

- Trabajo en equipo
- Habilidad para comunicar y transmitir procedimientos y resultados en base a razonamiento crítico

■ Competencias específicas:

- Reconocer las posibilidades que ofrece el uso sistemático de técnicas de extracción de conocimiento.
- Capacidad para sintetizar una técnica de aprendizaje automático no-supervisado, conocer su coste computacional así como sus limitaciones de representación y de inteligibilidad
- Aplicación: minería de textos. Caracterizaciones de documentos en espacios de atributos diversos.

3. Material

■ PC con aplicación Weka y SO Linux

■ Recursos accesibles desde e-Gela:

- Manual de la aplicación
- Bibliografía
- Conjuntos de muestras:
 - Carpetas de datos para el proyecto (incluyen un **readme** que contiene la descripción de los datos): [News](#), [Reuters](#), [Wikipedia](#), [Spam](#), [filmReviews](#), [tweetSentiment](#)
 - Ficheros para comenzar a familiarizarse con la tarea: [food.arff](#), [colon.arff](#), [landsat.arff](#), [amazonBoW.arff](#) [ClusterData.atributos.txt](#), [bank-data.csv](#).

Para profundizar en este contexto se sugieren los siguientes recursos: Minería de textos - marco teórico: *text mining* W. Richert, L. P. Coelho, *Building Machine Learning Systems with Python*. PACKT. July 2013. [Richert and Coelho, 2013].

4. Metodología de trabajo

La metodología de trabajo incluye los siguientes aspectos:

■ Naturaleza: diseño e implementación de software

- **Carácter:** grupal con grupos reducidos (3 personas, excepcionalmente 4)
- **Gestión del grupo:**
 - El propio grupo define unas reglas elementales para garantizar el buen funcionamiento del mismo y cada miembro adquiere el compromiso de cumplirlas. Este documento se subirá a e-Gela con el nombre: **Compromisos.pdf** antes de la fecha indicada en e-Gela. Al finalizar la tarea cada miembro del grupo se encargará de hacer una evaluación intra-grupal anónima.
 - Cada miembro del grupo contribuirá a que el resto alcance las competencias en las que destaca.
- **Asignación:** Cada grupo abordará una tarea distinta (especificada en la sección 7) mediante un algoritmo asignado cuyas especificaciones mínimas se describen en las secciones 6.1-6.2. El grupo podrá añadir otras funcionalidades que considere oportunas. Se pide pre-procesar los datos (según una de las técnicas que se indica en la sección 5), elige uno de los algoritmos de clustering (k-means, hierarchical) e impleméntalo sin utilizar librerías de MD externas (e.g. Weka).
- **Validación:**
 - Cada grupo diseñará una aplicación realista. La aplicación final se pondrá a disposición del resto de los grupos en la plataforma e-Gela.
 - El grupo dará garantías de que se han alcanzado los objetivos del proyecto mostrando el funcionamiento de su aplicación en una presentación oral.
 - Cada miembro validará las aplicaciones desarrolladas por los demás grupos en una evaluación inter-grupal.

5. Pre-proceso

Para este apartado se permiten utilizar librerías externas e.g. Weka [Witten et al., 2011, Sec. 6.8 y Sec. 11.6].

5.1. Formato

Los datos se han descargado de páginas web, y por lo tanto decimos que están en formato *raw*. El primer paso del pre-procesado consiste en adecuarlos a la herramienta con la que vamos a trabajar. En este caso, diseñaremos e implementaremos un software (**getARFF.jar**) que convierta el conjunto de datos al formato **.arff** (o, alternativamente, a **.csv**, ...). Dado el conjunto **data** asignado al grupo, obtendremos el fichero **data.arff**. Para este ejecutable, en caso de que no introduzcamos ningún argumento, se imprimirá por pantalla información útil sobre su funcionamiento (objetivo, argumentos y valor de retorno).

```
java -jar getARFF.jar data data.arff
```

Para el caso concreto de minería de textos, típicamente disponemos de textos en una carpeta `data` y para poder procesarlos con Weka, necesitamos reunirlos en un único fichero `data.arff`. Al aplicar la rutina de formateo (`getARFF.jar`) cada texto (mail, tweet,...) representa una instancia del fichero `data.arff` y se caracteriza mediante un único atributo (denominado `text` y de tipo `string`) que, en las tareas de clasificación tendría asociado un atributo adicional clase (denominado `class`) (pero no en tareas de *clustering*). En esta representación preliminar conviene ofrecer los datos filtrados siguiendo los siguientes criterios:

- *Encoding*: Filtrar caracteres que no siguen la codificación soportada por las librerías que utilizaremos después.
- *Tokenization*: Separar las palabras de los signos de puntuación y eliminar blancos redundantes. De ahí en adelante se considera “palabra” a cada token, es decir, cada elemento delimitado por espacios, comienzo o salto de línea).
- *Re-casing*: Pasar a minúsculas. Esto incrementa la frecuencia de aparición de los tokens y ayuda a los algoritmos basados en datos a encontrar regularidades.

Las librerías NLTK de python hacen muy sencilla esta tarea como se muestra en la figura 1.

```
1 from nltk import word_tokenize
2 from nltk.stem.porter import PorterStemmer
3 import re
4 from nltk.corpus import stopwords
5
6 cachedStopWords = stopwords.words("english")
7
8 def tokenize(text):
9     min_length = 3
10    words = map(lambda word: word.lower(), word_tokenize(text));
11    words = [word for word in words
12             if word not in cachedStopWords]
13    tokens = (list(map(lambda token: PorterStemmer().stem(token),
14                       words)));
15    p = re.compile('[a-zA-Z]+');
16    filtered_tokens =
17    list(filter(lambda token:
18               p.match(token) and len(token)>=min_length,
19               tokens));
20    return filtered_tokens
21
```

Figura 1: Python: lowercase, tokenize, stem.

Ejercicio 1 *Análisis del conjunto de datos:* Es crucial un análisis exhaustivo (cualitativo y cuantitativo) de la tarea. Se describe el objetivo de la tarea y a continuación la caracterización de los datos de partida. Para la caracterización se ofrece, al menos, los siguientes valores, siempre que sea posible, en forma tabular:

- *Número de instancias.* ¿Hay instancias repetidas?

- *Número y tipo de atributos.*
- *Rango de valores de los atributos (valores distintos, mínimo, máximo, mediana, etc.).*
- *¿Hay instancias con atributos missing)?*

El siguiente comando aporta información útil para caracterizar los datos (hay otros, puedes encontrarlos en la Wiki de Weka):

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

5.2. Espacio de decisión

En minería de textos se recurre, típicamente a un espacio de representación de tipo *Bag of Words* (BoW) o a su refinamiento TF-IDF, descritos en las secciones 5.2.1-5.2.2. La tendencia actual consiste en representar cada palabra en un espacio vectorial \mathbb{R}^n (*word embedding*) y después representar el texto completo mediante operaciones algebraicas en ese espacio mediante técnicas conocidas como *Deep Learning* que mencionaremos en la sección 5.2.3.

5.2.1. Bag of Words

El marco teórico de la representación BoW es conocida, con todo, se puede consultar en el apéndice B. En este apartado nos ceñiremos a detalles relacionados con el marco experimental.

Se desea convertir el atributo que recoge el mensaje de texto en un vector numérico, para ello se aplicará el filtro `weka.filters.unsupervised.attribute.StringToWordVector` [Witten et al., 2011, Sec. 6.8 y Sec. 11.6]. Este filtro convierte un atributo de tipo `String` en un conjunto de atributos que representan la presencia (o ausencia) 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

Si hasta este punto no se ha llevado a cabo ningún pre-proceso, puede resultar de gran interés considerar idénticas las palabras escritas en mayúsculas o en minúsculas estableciendo la opción:

lowerCaseTokens=True. En este apartado tanto TFTransform como IDTransform se establecen a False. En la Figura 11, se muestra de forma gráfica el valor de los parámetros mencionados (sin embargo, en la práctica se establecerán en el propio programa, no gráficamente).

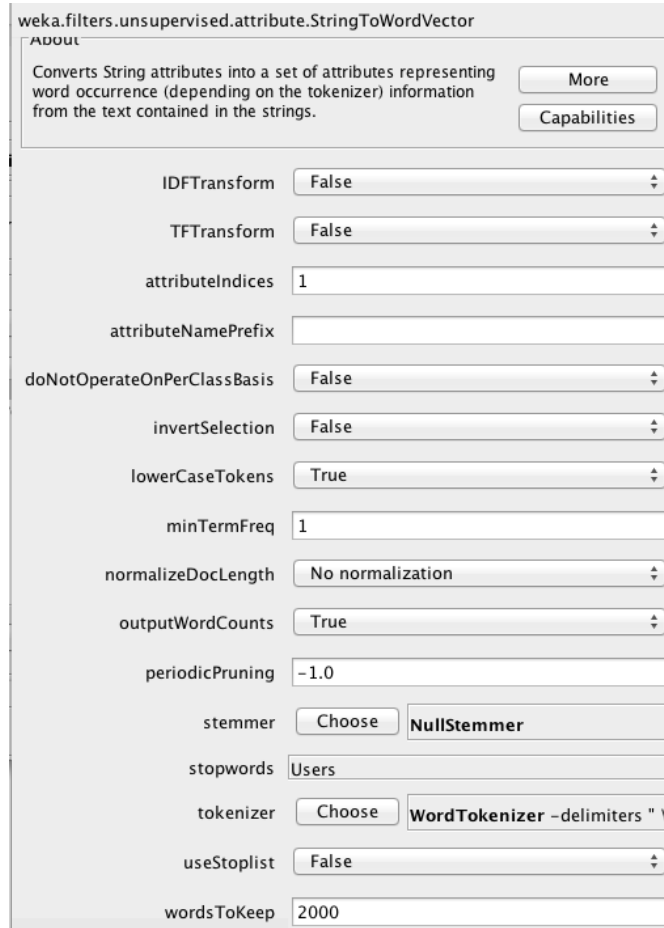


Figura 2: Transformar atributos de tipo String en BoW

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.

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 2 *Sparse ARFF*:

Al aplicar el filtro *StringToWordVector* este filtro sobre un conjunto de datos se obtiene como resultado una matriz dispersa: Sparse ARFF: <http://weka.wikispaces.com/ARFF>.

1. Define matriz dispersa con ayuda de <http://weka.wikispaces.com/ARFF>.
2. ¿Todos los algoritmos admiten matrices dispersas y no dispersas indistintamente?
3. La matriz dispersa se puede convertir en no-dispersa mediante el filtro *SparseToNonSparse* *weka.filters.unsupervised.instance*. Se pide aplicarlo para ver el resultado y describir, brevemente, la diferencia entre las dos matrices.

5.2.2. TF-IDF

Para recordar el marco teórico de la representación TF-IDF se puede consultar en el apéndice C. En este apartado nos ceñiremos a detalles relacionados con el marco experimental.

Para representar los textos mediante TF-IDF en Weka se puede encontrar en el filtro *StringToWordVector* (ya se utilizó en la sección 5.2.1), 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*).

Se pide diseñar e implementar software (*fssTFIDF.jar*) para cambiar la representación del fichero de datos *train.arff* a formato TFIDF (*trainBOW_FSS_TFIDF.arff*)

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

Ejercicio 3 *BoW vs TFIDF*:

1. Analiza el fichero *trainBOW_FSS_TFIDF.arff* y describe la representación de los datos. ¿cuántos atributos hay? ¿de qué tipo son? ¿qué representan?.
2. Explica, brevemente, la diferencia entre BoW y TFIDF.

5.2.3. Word Embeddings

Las representaciones de texto en formato BoW o TF-IDF emplean un número muy alto de atributos, del orden de la talla del vocabulario ($|\Sigma|$), incluso después de reducir la dimensión del espacio de decisión mediante técnicas de selección de variables. Estas técnicas tratan las palabras como símbolos discretos atómicos (e.g. para la palabra “gato” se le puede asignar el código *Id537* y la palabra “perro” *Id143* ambas de forma arbitraria, sin ninguna relación entre sí). Esta representación arbitraria no aporta información. Más aún, esta representación discreta de las palabras conlleva una gran dispersión en los datos. Dado que el número de identificadores es muy alto, la repetitividad de combinaciones de ellos se torna muy baja para un conjunto de datos dado.

Hay una representación alternativa emergente en investigación que trata de embeber una palabra en un vector \mathbb{R}^n , se conoce como *word-embedding*. En la figura 3 se muestra un ejemplo de la representación vectorial donde las palabras *fingerprint*, *hands*, *thumb*, *etc.* están muy próximas entre sí (cluster rojo), también están cerca entre sí y alejadas de las anteriores las palabras *sweet potato*, *potatoes*, *onion*, *asparagus*, *etc.* (cluster rosa), y algo similar sucede entre las palabras *tiger*, *panther*, *jaguar*, *etc.* (cluster azul).

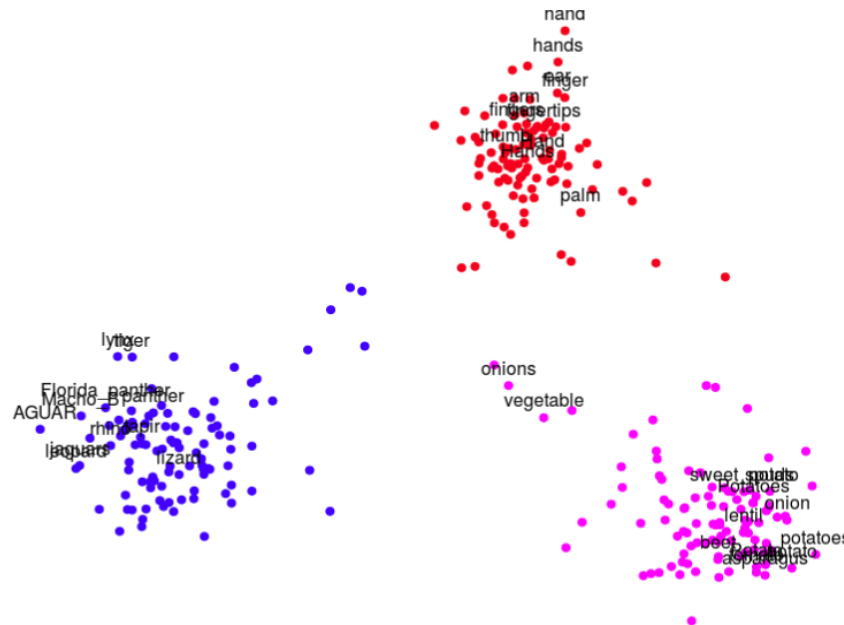


Figura 3: Palabras representadas en \mathbb{R}^2 .

Esta representación tiene dos notables ventajas frente a las representaciones clásicas BoW o TF-IDF:

- Típicamente, la dimensión del espacio suele ser $n = 200$, notablemente menor a la talla del vocabulario involucrado ($n \ll |\Sigma|$).
- Permite hacer operaciones algebraicas con los vectores asociados a las palabras. Así pues, este espacio es apropiado para buscar relaciones entre palabras mediante combinaciones lineales de vectores como se muestra en la figura 4. Por ejemplo, en base a la figura 4a descubrimos que operaciones elementales con vectores nos llevan a relaciones del tipo:

$$\overrightarrow{queen} - \overrightarrow{king} \approx \overrightarrow{woman} - \overrightarrow{man} \implies \overrightarrow{queen} \approx \overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman}$$

$$\overrightarrow{????} \approx \overrightarrow{uncle} - \overrightarrow{man} + \overrightarrow{woman}$$

Esta representación se obtiene mediante métodos basados en *Deep Learning* [LeCun et al., 2015]. Son modelos que tratan de imitar las redes neuronales de nuestro cerebro que lejos de ser un

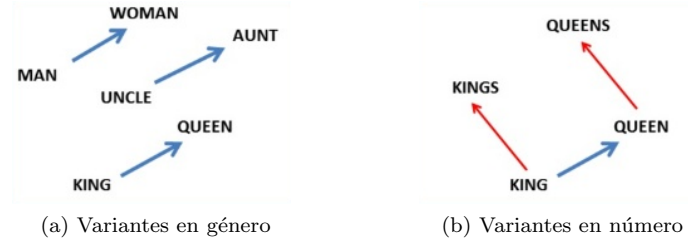


Figura 4: Relaciones entre palabras representadas en \mathbb{R}^2 .

modelo de dos capas ($input(features) \rightarrow output$) involucra múltiples niveles de procesamiento. Ejemplo: por lo que se conoce, el modelo del cortex para interpretar imágenes primero extrae bordes, después parches de color, identifica superficies y finalmente objetos [Murphy, 2012].

Concretamente, para la representación de palabras mediante vectores es común seguir la estructura mostrada en la figura 5. De forma superficial, dado un conjunto de texto grande no-supervisado, se entrena una estructura neuronal con distintas capas ocultas y se ajustan sus parámetros de modo teniendo como objetivo predecir los contextos de una palabra dada. Los parámetros (w_i) de la red sirven para representar las palabras en el espacio \mathbb{R}^n . La idea subyacente consiste en que palabras en similares contextos estarán localizadas en zonas similares del espacio.

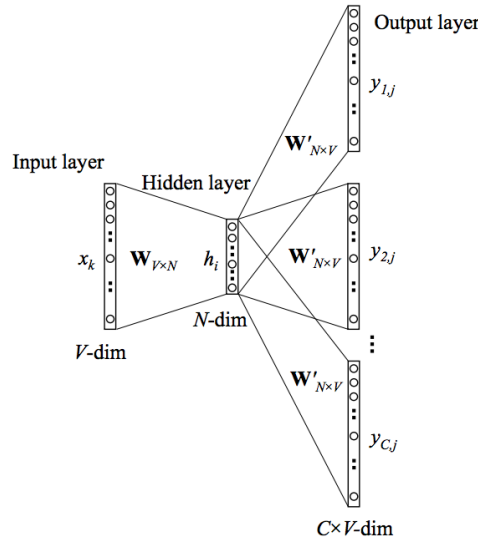


Figura 5: Modelo Skipgram para obtener *word-embeddings*.

Según la semántica distribucional (*distributional semantics*), palabras (o secuencias de palabras) que presentan una distribución similar tienen significados similares. Según el uso de una palabra se puede derivar su significado, en tanto que palabras que ocurren en un contexto simi-

lar tienden a ofrecer significados similares. Esta hipótesis de la semántica se puede está siendo explorada actualmente mediante *word-embeddings*. Entre las aplicaciones más destacables está el clustering de palabras basadas en similitud semántica. Hasta ahora apenas se conocían herramientas para medir la similitud semántica de las palabras. Con la representación vectorial de las palabras se abre un marco de trabajo para cuantificar usos de palabras en textos históricos, su evolución temporal, su relación con palabras en otros idiomas etc. No sólo se puede explorar a nivel de palabra, sino que abre un amplio abanico de posibilidades para representar un texto de forma vectorial.

Se ha documentado una serie de recursos sobre *deep learning* en el sitio web: deeplearning.net. Destacamos:

- **word2vec**: librería en C disponible en <https://code.google.com/archive/p/word2vec/>
- **keras**: es una API de Python que permite programación rápida de prototipos mediante interfaz amigable <https://keras.io/>
- **gensim**: librería para Python disponible en <http://radimrehurek.com/gensim/index.html>
 - **TensorFlow**: librería para computar datos numéricos utilizando gráficos de flujo de datos, especialmente indicada para trabajar con *deep neural networks* <https://www.tensorflow.org/versions/r0.11/tutorials/word2vec/index.html>
- **Deeplearning4j**: implementación en Java disponible en <http://deeplearning4j.org/word2vec>

5.3. Filtrado

Cuando se hace clustering con atributos numéricos reales (\mathbb{R}) típicamente se emplean los siguientes filtros:

- Normalización: `weka.filters.unsupervised.attribute.Normalize`
- Eliminar atributos que apenas varíen: `weka.filters.unsupervised.attribute.RemoveUseless`
- Eliminar instancias *outliers* requiere de la aplicación de 3 filtros:
 - `weka.filters.unsupervised.attribute.InterquartileRange`
 - `weka.filters.unsupervised.instance.RemoveWithValues`
 - `weka.filters.unsupervised.attribute.Remove`

Ejercicio 4 Filtrado: *Discute si para la tarea y la representación concreta que habéis hecho en vuestro grupo es necesario aplicar algún filtro (sea los que se han mencionado en la sección 5.3 u otro).*

6. Clustering

El trabajo creativo propio de implementación se centra en esta sección, se pide elegir uno de los algoritmos de inferencia de clustering (k-means, hierarchical) e implementarlo **sin utilizar librerías de MD externas** (e.g. Weka). Además, se pedirá implementar software para dar una estimación de la calidad del modelo y, un último paquete para aplicar este modelo sobre muestras no vistas en el entrenamiento.

6.1. Hierarchical clustering

Se implementará el algoritmo *bottom-up hierarchical clustering* [Alpaydin, 2010] que permita:

1. Seleccionar cualquier distancia de Minkowski ($\forall \alpha \in \mathbb{R}$) para calcular la distancia entre instancias. Sean $\mathbf{x} = (x_1, \dots, x_i, \dots, x_n) \in \mathbb{R}^n$ y $\mathbf{z} = (z_1, \dots, z_i, \dots, z_n) \in \mathbb{R}^n$, entonces, la distancia de Minkowski de grado α entre \mathbf{z} y \mathbf{x} se calcula según (1).

$$d(\mathbf{z}, \mathbf{x}) = \left[\sum_{i=1}^n |z_i - x_i|^\alpha \right]^{\frac{1}{\alpha}} \quad (1)$$

2. Seleccionar distancia inter-grupal:

- *Mean-link*: distancia media entre todos los pares de instancias de los dos grupos

$$d(\mathcal{G}_i, \mathcal{G}_j) = \frac{1}{|\mathcal{G}_i||\mathcal{G}_j|} \sum_{\mathbf{x}^{(r)} \in \mathcal{G}_i} \sum_{\mathbf{x}^{(s)} \in \mathcal{G}_j} d(\mathbf{x}^{(r)}, \mathbf{x}^{(s)}) \quad (2)$$

- *Single-link*:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \min_{\forall \mathbf{x}^{(r)} \in \mathcal{G}_i, \forall \mathbf{x}^{(s)} \in \mathcal{G}_j} d(\mathbf{x}^{(r)}, \mathbf{x}^{(s)}) \quad (3)$$

- *Complete-link*:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \max_{\forall \mathbf{x}^{(r)} \in \mathcal{G}_i, \forall \mathbf{x}^{(s)} \in \mathcal{G}_j} d(\mathbf{x}^{(r)}, \mathbf{x}^{(s)}) \quad (4)$$

- *Average-link*: distancia entre centroides

$$d(\mathcal{G}_i, \mathcal{G}_j) = d(\mathbf{m}_i, \mathbf{m}_j) \quad \text{donde } \mathbf{m}_i = \frac{1}{|\mathcal{G}_i|} \sum_{\forall \mathbf{x}^{(r)} \in \mathcal{G}_i} \mathbf{x}^{(r)} \quad (5)$$

3. Opcional: mostrar el dendograma resultante de forma gráfica. Para este paso se puede recurrir a librerías externas para representarlo (e.g. Weka, R, Python, ...).
4. Como resultado, el programa retornará:
 - Un fichero con distancia y clusters donde se muestre el resultado de cada iteración por fila (e.g. iteration=k; dig=3.7; {1,3,4,7}; {2,6}; {5,8,9})
 - Indicadores internos de la calidad del resultado (e.g. *Silhouette*, SSE, ...) para cada nivel del cluster (e.g. en la fila j del fichero debería haber, al menos, dos valores: NumClusters=j; Silhouette=sj).

5. Incluirá una librería adicional para explorar el fichero de salida y extraer información del agrupamiento:
 - Los clusters que se forman con distancia inter-grupal menor a una dada como parámetro. Para una distancia dada, ofrecer un fichero con la asignación de las instancias a los clusters: en la fila i indicará a qué cluster pertenece la instancia i y a continuación dará el vector de atributos de esa instancia.
 - Dado un número de clusters retornar los centroides correspondientes a la agrupación.
 - Dado un número de clusters y un conjunto de instancias (distintas al conjunto que se empleó para la partición), asignar a cada instancia el cluster al que pertenece. Es decir, el número de clusters sirve para determinar la agrupación de referencia que se utilizará y con esa agrupación (tomando sus centroides como referencia) indicar a qué cluster pertenecerían las nuevas instancias dadas.

6.2. *k-means clustering*

Se implementará el algoritmo *k-means clustering* [Alpaydin, 2010] que permita:

1. Modificar inicializaciones:
 - a) Aleatoria
 - b) Por división del espacio
 - c) Generar $2k$ clusters y de sus centroides elegir los k que estén más separados entre sí. Estos k centroides servirán para inicializar los k centroides del algoritmo *k-means clustering*.
2. Seleccionar cualquier distancia de Minkowski ($\forall m \in \mathbb{R}$) para calcular la distancia entre instancias según la expresión (1).
3. Seleccionar distancia inter-grupal:
 - Single-link según la expresión (3).
 - Complete-link según la expresión (2).
4. Modificar criterios de convergencia:
 - a) Un número de iteraciones fijo (especificado como una constante)
 - b) Disimilitud entre codebooks sucesivos menor a un umbral prefijado (especificado como una constante)
5. Como resultado, el programa retornará:
 - Las coordenadas de los centroides, es decir, el vector de atributos de los centroides finales.
 - Un fichero con la asignación de las instancias a los clusters: en la fila i indicará a qué cluster pertenece la instancia i y a continuación dará el vector de atributos de esa instancia.
 - Indicadores internos de la calidad del resultado (e.g. *Silhouette*, SSE).

6.3. Índices de calidad

En función de la tarea asignada y, concretamente, en función de si cuenta o no con clases pre-definida se implementarán índices de calidad internos sólo o internos y externos [Bandyopadhyay and Saha, 2013, Chapters 1, 4, 6].

Se valorará que el algoritmo indique el número óptimo de clusters en base a distintos criterios (e.g. Silhouette u otros).

Adicionalmente, se valorará una representación gráfica bidimensional sobre los 2 atributos más relevantes obtenidos mediante **PCA** donde se muestre la agrupación de las instancias y los centroides de cada cluster en cada iteración (como se indica en el apéndice A.2.2).

6.4. Aplicación del modelo inferido

En la sección anterior se ha inferido una agrupación. A continuación, vamos a aplicar esa agrupación para extraer información de otras instancias que no se han utilizado para entrenar el modelo. Vamos a proporcionar información a dos niveles:

1. Una vez asignados los clusters a cada instancia del conjunto original con el que se ha entrenado el modelo ¿cómo indicar a qué cluster pertenecería una nueva instancia (*query*).
 - Representar la nueva instancia en base a los atributos que se utilizaron para obtener el cluster.
 - Comparar el vector de atributos de la nueva instancia con los centroides obtenidos y asociar la *query* con el centroide más cercano.
2. Como resultado de aplicación, en el caso de cluster de noticias, en este punto hemos averiguado con qué cluster se relaciona la noticia *query*. A continuación podríamos mostrarle al usuario noticias relacionadas con una que está leyendo. En el paso anterior se ha asignado un cluster a la nueva *query*, deseamos mostrarle al usuario instancias relacionadas:
 - Ordenar los posts del conjunto inicial que estaban asociados al cluster del documento query por cercanía al post query
 - Mostrar sólo los N documentos más relevantes (donde N será un parámetro pre-seleccionable por el usuario). Mostrar el documento junto con su similitud.

7. Casos prácticos de aplicación

Nos centramos en minería de texto con las aplicaciones agrupar noticias relacionadas, buscar posts que tratan sobre el mismo tema, agrupar palabras según su significado en un dominio. Cada una de ellas viene descrita en su carpeta de datos correspondiente. A continuación se ofrece un resumen.

7.1. Clustering de posts

- **Data set:** UCI Machine Learning Repository

- <https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups>

Consta de un conjunto de ≈ 20.000 mails de 20 foros.

- **Objetivo:** En nuestro caso nos cuestionamos si un algoritmo de clustering (sin más supervisión) sería capaz de descubrir temas relacionados (agrupar los mensajes por temas). Dado un mensaje (e.g. una consulta), deseamos encontrar mensajes relacionados (e.g. temas similares para encontrar una respuesta a nuestra consulta). Si bien el conjunto original contiene el grupo al que pertenece, no debemos utilizarlo para hacer el clustering, sólo sirve para comprobar si el clustering propuesto está relacionado con los temas pre-seleccionados. También hay otro tipo de información sensible que los estudiantes deberían valorar si incluir o no: e.g. *Sender*, *Subject*, *Signature*... Observación: **No se puede utilizar la clase como atributo del clustering**, pero sí puede servir para hacer una evaluación externa. Por otra parte, la categorización no es excluyente, sino que puede haber múltiples clases asociadas a un mismo documento.

Explora si el agrupamiento resultante (el clustering propuesto con el algoritmo implementado en el equipo) está relacionado con la partición original. ¿Se podría utilizar algún índice externo (e.g. Jaccard, Rand-index, etc.) para esta evaluación? ¿Presenta correlación el cluster con respecto a la clase?

7.2. Reuters

- **Data set:** UCI Machine Learning Repository

- <https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection>

22 ficheros SGML. Cada uno de ellos comprende un conjunto de aproximadamente 1.000 documentos que se publicaron en *Reuters Newswire*.

- **Objetivo:** Es una tarea conocida y sobre la que se han publicado numerosos artículos científicos. Cuenta con múltiples etiquetas o clases, pero no se puede utilizar la clase como atributo del clustering. Sí puede servir para hacer una evaluación externa. Por otra parte, la categorización no es excluyente, sino que puede haber múltiples clases asociadas a un mismo documento. Es una tarea muy completa y bien documentada, así pues, no es necesario ofrecer más información en esta sección.

7.3. Clustering semántico

- **Data set:** Wikipedia (**9GB**)

- <http://dumps.wikimedia.org>

- Alternativamente se puede trabajar con otros conjuntos de datos pero hay que recordar que estos algoritmos requieren de grandes volúmenes de datos para ofrecer resultados competitivos. Se pueden encontrar alternativas en la sección “*Where to obtain the training data*” en <https://code.google.com/archive/p/word2vec/>
- **Objetivo:** Mediante la representación vectorial de las palabras se abre un marco de trabajo en el que medir cuantitativamente la similitud semántica de las palabras, las frases y los documentos. Se pide agrupar, automáticamente, palabras relacionadas. Éste es un campo ambicioso de la inteligencia artificial: ¿Se pueden descubrir otro tipo de relaciones entre palabras? Si sabemos cuál es la capital del país A ¿podríamos averiguar la capital del país B? Si sabemos la forma en singular de una palabra ¿podemos averiguar su forma en plural aunque no sea regular?

El software relacionado (mencionado en la página 11) y particularmente **word2vec** no solo es capaz de obtener las representaciones vectoriales de las palabras de un texto. También permite agrupar las palabras en clusters mediante el algoritmo k-means. Se pide implementar el algoritmo k-means y comparar los resultados con los clusters de **word2vec**. Para hacer la evaluación se utilizarán índices externos (e.g. Rand-index etc.). ¿Correla la agrupación implementada en el grupo con el cluster de **word2vec**? ¿Se podría utilizar algún índice externo (e.g. Jaccard, Rand-index, etc.) para esta evaluación?

7.4. Otras

Hay otras aplicaciones relacionadas con la minería de texto que se pueden explorar mediante clustering (están disponibles en la carpeta de datos):

- Detección de spam: **Spam**
- Polaridad en críticas de películas: **filmReviews**
- Sentiment analysis en tweets: **tweetSentiment**

8. Resultados

8.1. Entregables

Se pide un paquete con el código fuente diseñado y debidamente documentado (**src.gzip**) así como un informe del proyecto en formato PDF (**InformeP2.pdf**) que incluya las siguientes secciones:

1. Introducción:

- a) **Definición:** clasificación no-supervisada
- b) **Objetivo:** definir el problema concreto a abordar en el contexto de la minería de datos. Incluir, además, los siguientes detalles:
 - Instancias: ¿de cuántas instancias se dispone? ($N = \quad$)
 - Atributos: ¿cuántos atributos se emplean para describir las instancias? ($n = \quad$)
¿de qué tipo son?

2. Algoritmo: en pseudo-código

3. Diseño: mapa de diseño donde se muestran las dependencias y se documentan las rutinas

4. Resultados experimentales:

- a) Describir banco de pruebas diseñado para validar el software y resultados obtenidos
- b) Sobre los datos de aplicación de la tarea se pide obtener los siguientes resultados:
 - 1) Distinto número de clusters a considerar (eg. k : 2, 3, 12)
 - 2) Distintas métricas:
 - Distancia de Manhattan ($m = 1$)
 - Distancia Euclídea ($m = 2$)
 - Distancia de Minkowski con $m = 7.5$
 - 3) Distintas inicializaciones
 - 4) Distintos criterios de convergencia
- c) **Análisis crítico y discusión de resultados**
- d) **Rendimiento del software:** discutir el rendimiento del software diseñado en términos de coste temporal real y espacio en memoria requerido. Dar el perfil de tiempo y memoria consumido por cada modelo explorado. Comparar estos resultados con los que se obtienen con Weka.

5. Conclusiones: argumentar las siguientes cuestiones.

- a) Describir muy brevemente la motivación para llevar a cabo técnicas de *clustering*
- b) Describir muy brevemente las conclusiones obtenidas a la vista de los resultados más relevantes
- c) Conclusiones generales (análisis de fortalezas del software implementado y reflexiones más importantes sobre la tarea)
- d) Propuestas para mejorar o ampliar la funcionalidad del software diseñado en trabajo futuro (análisis de puntos débiles y propuesta para solucionarlos)

6. **Bibliografía:** se cita la fuente empleada en el punto en el que se recurre a ella para fundamentar una argumentación (detallando la página o sección). En esta última sección del informe se detalla la fuente de las fuentes citadas (y sólo esas, es decir, no aparecerán más referencias que las citadas). Deben aparecer tantos detalles sobre la fuente como sean posibles, y al menos: título, autor, editorial, año.

8.2. Evaluación

Se valorarán los siguientes aspectos:

1. Software:

- a) Algoritmo y Diseño del software
- b) Implementación:
 - Modularidad
 - Portabilidad
 - Eficiencia
 - Documentación de cada rutina (recordar metodología de la programación: pre- y post- condiciones)
- c) Documentación del paquete de software: descripción del modo de uso del software (documentación tipo `leeme.txt`) donde se indica, entre otros, modo de compilación (empleando `Makefile`) y ejemplo de ejecución
- d) Innovación: el proyecto debería ir más allá de los mínimos detallados en este guión, debería aportar un punto original, innovador (e.g. en términos de capacidades, representación, visualización, robustez, eficiencia, etc.).

2. Informe:

- a) Completitud
- b) Formalismo y fundamento teórico, expresión escrita en el ámbito científico-técnico
- c) Experimentación exhaustiva
- d) Análisis crítico de resultados
- e) Bibliografía
- f) Además del contenido, se valorará la presentación.

8.3. Plazos

Los plazos de entrega están indicados en e-Gela.

Referencias

- [Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. MIT Press.
- [Bandyopadhyay and Saha, 2013] Bandyopadhyay, S. and Saha, S. (2013). *Unsupervised Classification*. Springer.
- [LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.
- [Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- [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.

Apéndices

A. Imágenes tomadas vía satélite

A.1. Descripción de la tarea

- **Data set:** UCI Machine Learning Repository
 - [https://archive.ics.uci.edu/ml/datasets/Statlog+\(Landsat+Satellite\)](https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite))
- **Objetivo:** el objetivo original de la tarea era inferir un modelo predictor para clasificar superficies terrestres en los siguientes términos: 1. *red soil*; 2. *cotton crop*; 3. *grey soil*; 4. *damp grey soil*; 5. *soil with vegetation stubble*; 6. *mixture class (various types are present)* (no hay instancias de esta clase); 7. *very damp grey soil*. Así pues, disponemos de un conjunto de imágenes de la superficie terrestre tomada vía satélite junto con un atributo especial denotado por “class” que hace referencia a esta clasificación. Nuestro objetivo consistirá en descubrir si las técnicas de *clustering* ayudarían a describir la tarea de una forma eficaz. ¿Existe correlación entre los clusters que derivamos y la clase real? ¿El *cluster* podría ayudar como atributo predictor?
- **Descripción:** El conjunto de datos representa imágenes de la superficie terrestre tomadas vía satélite. De cada zona se han tomado imágenes en 4 bandas espectrales distintas (dos se corresponden, aproximadamente con el verde y el rojo de la región visible del espectro y otras dos cerca del infra-rojo) como se muestra en la figura 6. Cada muestra consiste en una imagen de 3×3 pixels donde cada pixel representa un área de la superficie terrestre de $80m \times 80m$. Cada pixel consistía en una secuencia binaria de 8 bits que ha sido transformada a base decimal y donde 0 representa el “negro” y 255 el “blanco”. En la tarea original de clasificación, la clase hace referencia al tipo de la superficie terrestre ($80m \times 80m$) asociado con el pixel central de la imagen (denotado por “px5” en la figura 6) y se considera que las regiones del colindantes pueden ayudar a predecir la clase de esa región. En el conjunto de datos los pixels se han dispuesto en atributos según se muestra en la figura 7, donde, además se ofrece el atributo clase.

| | | | | | | |
|--------|--------|--------|-----|--------|--------|--------|
| px1fr1 | px2fr1 | px3fr1 | ... | px1fr4 | px2fr4 | px3fr4 |
| px4fr1 | px5fr1 | px6fr1 | ... | px4fr4 | px5fr4 | px6fr4 |
| px7fr1 | px8fr1 | px9fr1 | ... | px7fr4 | px8fr4 | px9fr4 |

Figura 6: Caracterización de una región terrestre mediante una imagen multi-espectral: imagen de 3×3 pixels tomada en 4 frecuencias distintas.

| | | | | | | | | | | | |
|--------|--------|--------|--------|--------|-----|--------|-----|--------|-----|--------|-------|
| px1fr1 | px1fr2 | px1fr3 | px1fr4 | px2fr1 | ... | px5fr1 | ... | px5fr4 | ... | px9fr4 | class |
| atr1 | atr2 | atr3 | atr4 | atr5 | ... | atr17 | ... | atr20 | ... | atr36 | atr37 |

Figura 7: Disposición de pixels en atributos.

A.2. Procedimiento

A.2.1. Clustering

Ejercicio 5 Para empezar, exploar un clustering *k*-medias (*weka.clusterers.SimpleKMeans*) con tantos grupos como clases y evaluar el resultado de la agrupación con respecto de la clase. En la figura 8 se muestra un resultado preliminar de la evaluación de un 6-means clustering. Si

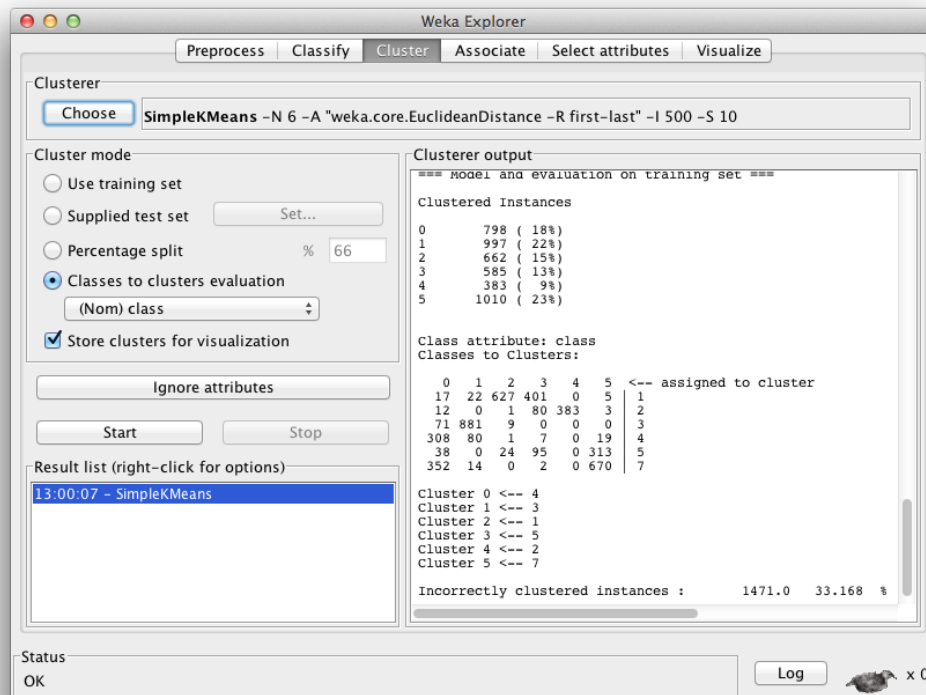


Figura 8: k-means clustering

bien las técnicas de clustering son técnicas exploratorias, se puede observar que la agrupación guarda cierta relación con la clase pre-definida:

1. ¿Cuál sería la tasa de acierto si pretendiésemos clasificar las muestras en base al modelo no-supervisado?
2. ¿Qué clases son las que más se confunden entre sí con esta agrupación?
3. ¿Cuál es la correlación del clustering respecto de la clase?
4. Mediante técnicas no-supervisada podemos asignar a cada instancia una característica, el cluster al que pertenece. ¿Se podría utilizar esta característica como atributo predictor? ¿Es un buen atributo predictor en esta tarea?

A.2.2. Representación PCA

Ejercicio 6 *Análisis de componentes principales* (PCA: Principal Component Analysis)

1. PCA permite obtener un nuevo conjunto de atributos mediante combinaciones lineales de los atributos originales ¿con qué propósito? ¿cómo obtienes esos atributos? En definitiva hemos hecho un cambio de base.
2. Podemos seleccionar atributos en esa nueva base como se muestra en la figura 10 (ahí se ha usado `weka.attributeSelection.PrincipalComponents`). Según esta técnica ¿cuántos atributos de este espacio son suficientes para aportar información relevante? Weka per-

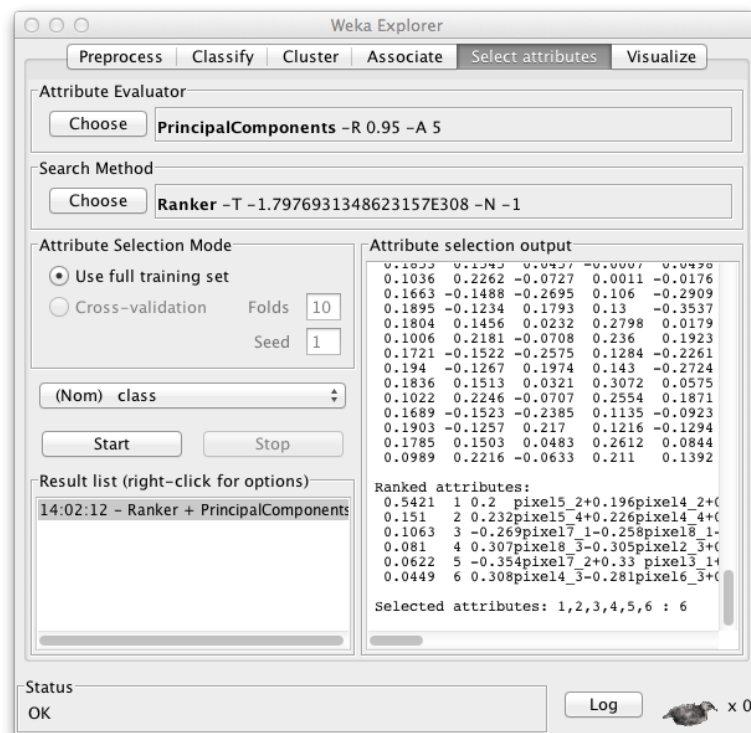


Figura 9: Ranking de atributos mediante PCA.

mite visualizar los datos transformados y también salvarlos. A la vista de la visualización ¿Parecen las clases más separables en el espacio PCA que en el espacio original?

3. Una vez filtrados los datos mediante la selección de atributos según PCA (figura 10) estamos representando los datos con un número notablemente menor de atributos. ¿Es posible obtener mejores resultados con menos atributos? ¿Perderemos información? Si volvemos a hacer el clustering en este espacio (ejercicio 5) podemos comprobarlo en un problema de clasificación no-supervisada. Podríamos estudiar también qué sucede en clasificación supervisada, por ejemplo, seleccionando un clasificador *Naive Bayes* y un esquema de evaluación 10-fold cross validation.

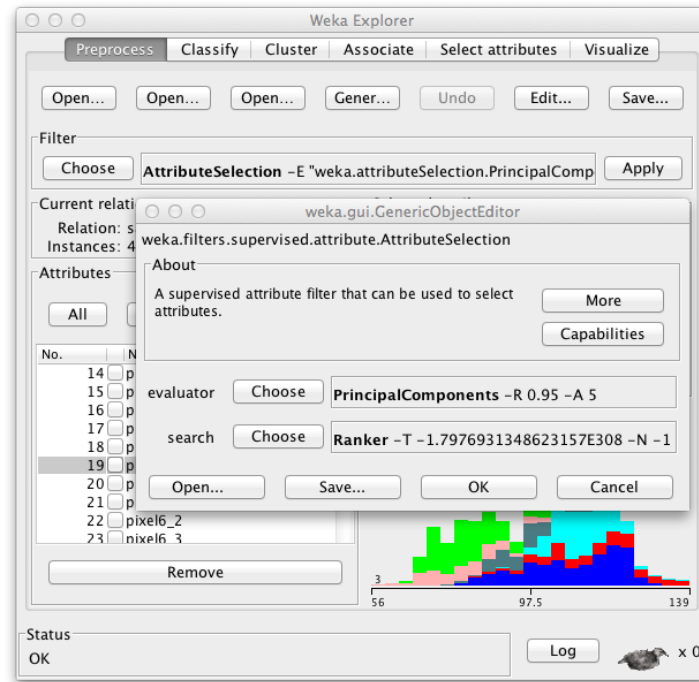


Figura 10: Pre-proceso con selección de atributos mediante PCA.

4. Se pide resumir para qué sirve PCA.

B. Marco teórico: Bag of Words

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 alternatively 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 11, se muestra de forma gráfica el valor de los parámetros mencionados (sin embargo, en la práctica se establecerán en el propio programa, no gráficamente).

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 dispersa (se puede convertir en no-dispersa mediante el filtro `weka.filters.unsupervised.instance.SparseToNonSparse`). Ayuda: *Sparse ARFF*: <http://weka.wikispaces.com/ARFF>

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

IDFTTransform

TFTTransform

attributeIndices

attributeNamePrefix

doNotOperateOnPerClassBasis

invertSelection

lowerCaseTokens

minTermFreq

normalizeDocLength

outputWordCounts

periodicPruning

stemmer

stopwords

tokenizer

useStoplist

wordsToKeep

Figura 11: Transformar atributos de tipo String en BoW

B.1. Dataset: Amazon

Como muestra, se ofrece un conjunto de datos que viene originalmente en formato BoW: [Amazon](#).

■ Data set: UCI Machine Learning Repository

- <https://archive.ics.uci.edu/ml/datasets/Amazon+Commerce+reviews+set>

Consiste en reseñas escritas por 50 usuarios del sitio web *Amazon Commerce*. Hay 30 reseñas de cada autor. El fichero está dado en formato BoW (información adicional en el apéndice B).

- **Objetivo:** este conjunto de datos se empleó originalmente para descubrir quién ha escrito algo en internet. En nuestro caso nos cuestionamos si los algoritmos de clustering podrían descubrir similitudes en las reseñas (quizá el estilo, la temática, etc.).

C. Marco teórico: TF-IDF

Se define la frecuencia relativa del término w_i en el documento d_j , *term frequency* (TF), según la expresión (6):

$$TF(w_i, d_j) = \frac{f(w_i, d_j)}{\sum_{w_i \in V} f(w_i, d_j)} \quad (6)$$

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 (7):

$$DF(w_i) = \frac{\sum_{d_j \in D} \delta(w_i, d_j)}{|D|} \quad (7)$$

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 (8).

$$\delta(w_i, d_j) = \begin{cases} 1 & w_i \text{ está presente en el documento } d_j \\ 0 & w_i \text{ no está presente en el documento } d_j \end{cases} \quad (8)$$

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

Así mismo, 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*)¹ según la expresión (9):

$$TFIDF(w_i, d_j) = TF(w_i, d_j) \cdot \log \frac{1}{DF(w_i)} \quad (9)$$

¹TF-IDF: ¡no denota la resta de las dos magnitudes! de hecho, está relacionada con el producto

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.

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](#) ▷²
- “Introduction to Information Retrieval”, C. Manning, P. Raghavan and H. Schütze, Cambridge University Press. 2008. [leer IR book](#) ▷³

C.1. Python: TF-IDF

La implementación de TF-IDF mediante librerías **sklearn** de Python comprende tres pasos:

1. Obtener representación TF-IDF: figura 12
2. Convertir TF-IDF a una representación amigable (*human friendly*): figura 13
3. Proyección de conjuntos externos en el espacio de decisión TF-IDF obtenido previamente: figura 14

```
1 # Return the representer, without transforming
2 def tfidf(docs):
3     tfidf = TfidfVectorizer(tokenizer=tokenize, min_df=3,
4                             max_df=0.90, max_features=3000,
5                             use_idf=True, sublinear_tf=True,
6                             norm='l2');
7     tfidf.fit(docs);
8     return tfidf;
9
```

Figura 12: Python: TF-IDF.

²<http://opencourseonline.com/273/stanford-university-nature-language-processing-video-playlist-19-ranked-information-retrieval>

³<http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.htm>

```
1 def main():
2     train_docs = []
3     test_docs = []
4
5     for doc_id in Reuters.fileids():
6         if doc_id.startswith("train"):
7             train_docs.append(Reuters.raw(doc_id))
8         else:
9             test_docs.append(Reuters.raw(doc_id))
10
11     representer = tf_idf(train_docs);
12
13     for doc in test_docs:
14         print(feature_values(doc, representer))
15
```

Figura 13: Python: TF-IDF convertido a un formato más amigable con nombre y peso de componentes no-nulas.

```
1 def main():
2     train_docs = []
3     test_docs = []
4
5     for doc_id in Reuters.fileids():
6         if doc_id.startswith("train"):
7             train_docs.append(Reuters.raw(doc_id))
8         else:
9             test_docs.append(Reuters.raw(doc_id))
10
11     representer = tf_idf(train_docs);
12
13     for doc in test_docs:
14         print(feature_values(doc, representer))
15
```

Figura 14: Python: TF-IDF compatible para el conjunto de test.