

Proyecto Text Mining

Predicciones

Participantes:

Paula de Jaime de Toro
Edgar Andrés Santamaría
Maitane Ruiz Monroy

Índice de contenidos

| | |
|--|-----------|
| Introducción a la minería de textos | 2 |
| Minería de textos y donde se aplica | 2 |
| Contextualizar la minería de textos en el ámbito de inteligencia artificial | 3 |
| Retos que entraña la minería de textos frente a otro tipo de datos | 3 |
| Pre-procesamiento | 5 |
| Características más relevantes de los tipos de datos raw, BoW o TFIDF | 5 |
| Transformaciones de los documentos | 5 |
| Manual de uso | 7 |
| Detalles relevantes del diseño y la implementación del software | 12 |
| Introducción experimental | 12 |
| Diseño | 13 |
| Implementación | 17 |
| Distribución de tareas | 20 |
| Selección de atributos | 20 |
| Clasificación | 21 |
| Baseline | 21 |
| Algoritmo asignado | 22 |
| Multilayer Perceptron | 22 |
| Diferencia entre la fase de inferencia del modelo y la fase de aplicación del modelo para hacer las predicciones | 25 |
| Marco experimental | 26 |
| Gráfico del funcionamiento de los software | 26 |
| Adquisición y pre-proceso de datos | 26 |
| Inferencia del clasificador | 27 |
| Predicciones | 28 |
| Pruebas realizadas | 29 |
| Adquisición y pre-proceso de datos | 29 |
| Inferencia del clasificador | 30 |
| Predicciones | 31 |
| Resultados experimentales | 33 |
| Espacio de representación | 36 |
| Aplicación de selección de atributos | 36 |
| Comparativa de modelos | 37 |
| Conclusiones | 38 |
| Referencias | 41 |

Introducción a la minería de textos¹

Minería de textos y donde se aplica

La minería de textos se refiere al conjunto de procesos que consiguen extraer significado (datos estructurados) de un conjunto de textos (datos no estructurados). Algunas tareas típicas de la minería de textos tenemos:

➤ **Categorización de textos** (*Text categorization/Document classification*)

Busca asignar un documento, de forma genérica, o un texto, de forma particular, a una o más categorías o clases.

- Se aplica en: filtrado de spam, identificación del idioma, análisis de sentimientos, etc.

➤ **Similitud semántica de textos** (*Semantic Text similarity*)

Captura el grado de similitud semántica de dos porciones de texto.

- Se aplica en: los buscadores de internet, dentro del campo de Búsqueda y Recuperación de Información.

➤ **Resumen de documentos** (*Document summarization*)

Intenta encontrar un conjunto de datos representativo de todo el documento, identificando para ello sus partes más informativas. El problema también se puede generalizar para hacer un resumen de múltiples documentos.

- Se aplica en: una de sus principales aplicaciones son los buscadores de internet.

➤ **Extracción de conceptos o entidades** (*Named entity recognition*)

Busca localizar y clasificar los elementos de un texto dentro de unas categorías predefinidas, como pueden ser nombres de personas, nombres de empresas, lugares, porcentajes, expresiones de tiempo, etc.

- Se aplica en: técnica es la identificación de entidades o expresiones importantes en los textos para crear vínculos directamente con Wikipedia 3 (entity linking)

➤ **Desambiguación del significado** (*Word Sense Disambiguation*)

Aborda el problema de seleccionar el sentido en el que una palabra es usada en una frase, cuando la palabra tiene múltiples significados (por ejemplo, banco puede referirse a banco de peces, al banco como entidad financiera, etc.)

- Se aplica en: los motores de búsqueda.

¹ [Bibliografía: Introducción a la minería de textos](#)

➤ ***Análisis de sentimientos (Sentiment Analysis/Opinion mining)***

Tiene como objetivo determinar la polaridad general de un documento, una frase o un aspecto del mismo, tratando de detectar la actitud del creador en base a las posibles emociones, juicios o evaluaciones contenidas en el documento.

- Se aplica en: las aplicaciones del Análisis de Sentimientos cobran especial relevancia, fundamentalmente desde el punto de vista empresarial, de cara a analizar todo tipo de expresiones online: análisis de revisiones y opiniones de productos, gestión de la reputación, identificación de nuevas oportunidades de negocio, etc.

Contextualizar la minería de textos en el ámbito de inteligencia artificial

Rama de la inteligencia artificial, en el contexto de la minería de datos, su objetivo es crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

El Procesamiento del Lenguaje Natural, como subrama de la Inteligencia Artificial y la Lingüística Computacional, cobra una relevancia clave, con el objetivo de extraer masivamente el significado residente en el lenguaje natural o humano.

Retos que entraña la minería de textos frente a otro tipo de datos

A día de hoy es un campo abierto con múltiples retos por resolver, al igual que el resto de ramas del Procesamiento del Lenguaje Natural con las que está íntimamente relacionado: identificar las entidades y atributos, clasificar sus polaridades, clasificar la polaridad general, etc.

Pre-procesamiento

Características más relevantes de los tipos de datos raw, BoW o TFIDF

Las características más relevantes se basan en los atributos que se diferencian de la siguiente forma:

- RAW ("text":String, class: nominal)
- BOW ("todas las palabras":numeric, class: nominal)
- TFIDF("palabras más representativas": numérico, class: nominal)

Transformaciones de los documentos

Los programas usados para realizar transformaciones son:

➤ **getRaw**

- Precondición: Ficheros a convertir pertenecen a uno de los conjuntos de datos siguientes:
 - movie_reviews
 - tweet_sentiment
 - sms_spam
- Post-condición: Fichero .arff se ha creado

Directories: java -jar getRaw.jar -d <pathorigen> <pathdestino.arff>

```
java -jar getRaw.jar -d /home/user/movies_reviews/train/ /home/user/movies_reviews/train.arff
java -jar getRaw.jar -d /home/user/movies_reviews/dev/ /home/user/movies_reviews/dev.arff
java -jar getRaw.jar -d /home/user/movies_reviews/test_blind/ /home/user/movies_reviews/test.arff
```

CSVs: java -jar getRaw.jar -c <pathorigen.csv> <pathdestino.arff> <carpeta>

```
java -jar getRaw.jar -c /home/user/tweet_sentiment_eGela/tweetSentiment.train
/home/user/tweet_sentiment_eGela/train.arff
java -jar getRaw.jar -c /home/user/tweet_sentiment_eGela/tweetSentiment.dev
/home/user/tweet_sentiment_eGela/dev.arff
java -jar getRaw.jar -c /home/user/tweet_sentiment_eGela/tweetSentiment.test_blind
/home/user/tweet_sentiment_eGela/test.arff
```

Texts: java -jar getRaw.jar -t <pathorigen.txt> <pathdestino.arff>

```
java -jar getRaw.jar -t /home/user/sms_spam/SMS_SpamCollection.train.txt
/home/user/movies_reviews/train.arff
java -jar getRaw.jar -t /home/user/sms_spam/SMS_SpamCollection.dev.txt /home/user/sms_spam/dev.arff
java -jar getRaw.jar -t /home/user/sms_spam/SMS_SpamCollection.test_blind.txt /home/user/sms_spam/test.arff
```

➤ transformRaw:

- Precondición: Los ficheros a convertir estan en formato raw.arff
- Post-condición: El fichero .arff, el diccionario y config.txt se han creado

```
java -jar transformRaw.jar <OPCION1> <OPCION2> <origen.arff> <destino.arff> <pathDiccionario>  
<config.txt>
```

| | |
|-----------|---|
| <OPCION1> | -tfidf (algoritmo TFIDF) - bow (algoritmo BOW) |
| <OPCION2> | -s (sparse) -ns (non-sparse) |

```
java -jar transformRaw.jar -bow -s /home/user/movies_reviews/train.arff  
/home/user/movies_reviews/trainBOW.arff /home/user/movies_reviews/diccionarioBOW config.txt
```

➤ makeCompatible:

- Precondición: Los ficheros a compatibilizar están en formato .arff y existe un diccionario del fichero train sobre el que compatibilizar
- Post-condición: El fichero .arff compatible se ha creado.

```
java -jar makeCompatible.jar <config.txt> <pathDiccionario> <origen.arff> <destino.arff>
```

```
java -jar makeCompatible.jar config.txt /home/user/movies_reviews/diccionarioBOW  
/home/user/movies_reviews/dev.arff /home/user/movies_reviews/devBOWCompatible.arff  
java -jar makeCompatible.jar config.txt /home/user/movies_reviews/diccionarioTFIDF  
/home/user/movies_reviews/dev.arff /home/user/movies_reviews/devTFIDFCompatible.arff
```

Para poder acceder a los resultados parciales de las transformaciones de cada unos de los programas anteriormente mencionados, se podrá acceder a los archivos “.arff”, mediante el siguiente enlace:

Enlace:

<https://drive.google.com/drive/folders/1Omi4gBCcn4kuAQazhW51p0Bpu209rRVT?usp=sharing>

Manual de uso

Se va a realizar todo el proceso mencionado anteriormente utilizando archivos con un número mínimo de instancias.

Los archivos a utilizar y los obtenidos en las siguientes instrucciones se pueden encontrar en el siguiente enlace:

Enlace: https://drive.google.com/drive/folders/1ToonS3vQJNG56qeSWN4_HIFT1LW4DE0g?usp=sharing

- Visualizamos los archivos .arff “train.arff” y “test.arff” (Fig. 1 y 2)

```
train.arff
@relation '_home_edgar_Universidad_2017-2018_2doCuatri_SAD-sistemas de apoyo a la decisión Practicas_pVoluntaria1_Datos_movies_reviews_train'

@attribute text string
@attribute @@class@@ {pos,neg}

@data
'hola a todos, me gusto. \n',pos
'hola, me disgusto. \n',neg
'a mi caesar, es perfecto. \n',pos
'buenos dias,no fue buen aporte . \n',neg
'gracias por el aporte. \n',pos
'buen trabajo. \n',pos
'no me gusto. \n',neg
```

Archivo train.arff (Fig. 1)

```
test.arff
@relation text_files_in_test_blind

@attribute text string
@attribute @@class@@ {neg,pos}

@data
'me gusto. \n',?
'mal aporte. \n',?
'perfecto. \n',?
'buenos dias,no fue buen aporte . \n',?
'gracias por el aporte. \n',?
'mal trabajo. \n',?
'me disgusto. \n',?
```

Archivo test.arff (Fig. 2)

- El primer paso sería aplicar el filtro ‘StringToWordVector’ al archivo ‘train.arff’ para lograr un fichero .arff con un espacio de atributos definido por las palabras encontradas en el atributo “text” de train.

Se utiliza el programa ‘transformRAW.jar’, configurando éste:

- BOW:

- Sparse

```
java -jar transformRaw.jar -bow -s train.arff BOW_SPARSE/train.arff BOW_SPARSE/diccionario
BOW_SPARSE/config.txt
```

```
@attribute disgusto numeric
@attribute fue numeric
@attribute no numeric

@data
{1 1,8 1,9 1,10 1,14 1}
{0 neg,9 1,10 1,18 1}
{1 1,4 1,6 1,11 1,12 1}
{0 neg,2 1,3 1,16 1,17 1,19 1,20 1}
{2 1,5 1,7 1,13 1}
{3 1,15 1}
{0 neg,8 1,10 1,20 1}
```

BOW_SPARSE/train.arff (Fig. 3)

➤ Non-sparse

```
java -jar transformRaw.jar -bow -ns train.arff BOW_NONSPARSE/train.arff
BOW_NONSPARSE/diccionario BOW_NONSPARSE/config.txt
```

```
@attribute fue numeric
@attribute no numeric

@data
pos,1,0,0,0,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0
neg,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0
pos,1,0,0,1,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0
neg,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1
pos,0,1,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0
pos,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
neg,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,1
```

BOW_NONSPARSE/train.arff (Fig. 4)

➤ TFIDF:

➤ Sparse

```
java -jar transformRaw.jar -tfidf -s train.arff TFIDF_SPARSE/train.arff TFIDF_SPARSE/diccionario
TFIDF_SPARSE/config.txt
```

```
2 @attribute fue numeric
3 @attribute no numeric
4
5 @data
6 {1 0.877499,8 0.877499,9 0.877499,10 0.593491,14 1.363014}
7 {0 neg,9 1.080705,10 0.730927,18 1.678653}
8 {1 0.651446,4 1.011888,6 1.011888,11 1.011888,12 1.011888}
9 {0 neg,2 0.664447,3 0.664447,16 1.032082,17 1.032082,19 1.032082,20
10 {2 0.740724,5 1.150562,7 1.150562,13 1.150562}
11 {3 1.150856,15 1.787619}
12 {0 neg,8 1.35622,10 0.917271,20 1.35622}
13
```

TFIDF_SPARSE/train.arff (Fig. 5)

➤ Non-Sparse

```
java -jar transformRaw.jar -tfidf -ns train.arff TFIDF_NONSPARSE/train.arff
TFIDF_NONSPARSE/diccionario TFIDF_NONSPARSE/config.txt
```

```
@attribute fue numeric
@attribute no numeric

@data
pos,0.877499,0,0,0,0,0,0.877499,0.877499,0.593491,0,0,1.363014,0,0,0,0,0
neg,0,0,0,0,0,0,0,1.080705,0.730927,0,0,0,0,0,1.678653,0,0
pos,0.651446,0,0,1.011888,0,1.011888,0,0,0,1.011888,1.011888,0,0,0,0,0,0
neg,0.664447,0.664447,0,0,0,0,0,0,0,0,0,0,0,1.032082,1.032082,0,1.032082,0.664447
pos,0.740724,0,0,1.150562,0,1.150562,0,0,0,0,0,1.150562,0,0,0,0,0,0
pos,0,0,1.150856,0,0,0,0,0,0,0,0,0,0,1.787619,0,0,0,0,0
neg,0,0,0,0,0,0,1.35622,0,0.917271,0,0,0,0,0,0,0,0,1.35622
```

TFIDF_NONSPARSE/train.arff (Fig. 6)

- Tras ejecutar las anteriores sentencias dispondremos de un archivo ‘diccionario’ que contendrá todas las palabras que el filtro ha detectado. (Fig.7) Además, se habrá guardado un archivo de configuración “config.txt” que establecerá en ocasiones futuras las opciones “sparse/non-sparse” y “bow/tfidf” elegidas en train.



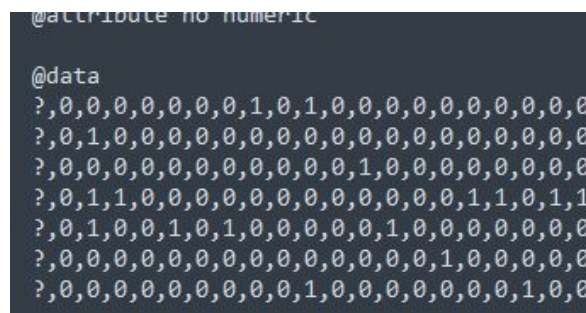
Diccionario ejemplo (Fig. 7)

- El segundo paso consistirá en compatibilizar el fichero “test.arff” utilizando el diccionario y archivo de configuración generados.

Utilizaremos el programa “*makeCompatible.jar*” de la siguiente manera:

```
java -jar makeCompatible.jar BOW_NONSPARSE/config.txt BOW_NONSPARSE/diccionario test.arff
BOW_NONSPARSE/test.arff
```

- ** Para hacer el fichero test.arff compatible a un fichero train.arff de tipo BOW y Non-Sparse, utilizaremos la configuración y el diccionario previamente generados al conseguir el fichero train BOW y Non-Sparse.**



BOW NONSPARE/test.arff (Fig. 8)

El resto de sentencias son las siguientes:

➤ BOW y SPARSE:

```
java -jar makeCompatible.jar BOW_SPARSE/config.txt BOW_SPARSE/diccionario test.arff
BOW_SPARSE/test.arff
```

```
@attribute no numeric

@data
{0 ?,8 1,10 1}
{0 ?,2 1}
{0 ?,12 1}
{0 ?,2 1,3 1,16 1,17 1,19 1,20 1}
{0 ?,2 1,5 1,7 1,13 1}
{0 ?,15 1}
{0 ?,10 1,18 1}
```

BOW_SPARSE/test.arff (Fig. 9)

➤ TFIDF y NONSPARSE

```
java -jar makeCompatible.jar TFIDF_NONSPARSE/config.txt TFIDF_NONSPARSE/diccionario test.arff
TFIDF_NONSPARSE/test.arff
```

```
@attribute no numeric
@attribute no numeric

@data
?,0,0,0,0,0,0,0,1.503338,0,1.503338,0,0,0,0,0,0,0,0,0
?,0,2.126041,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
?,0,0,0,0,0,0,0,0,0,0,2.126041,0,0,0,0,0,0,0,0
?,0,0.867953,0.867953,0,0,0,0,0,0,0,0,0,0,0.867953,0.867953,0,0.867953,0.867953
?,0,1.063021,0,0,1.063021,0,1.063021,0,0,0,0,0,1.063021,0,0,0,0,0,0
?,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2.126041,0,0,0,0,0
?,0,0,0,0,0,0,0,0,0,1.503338,0,0,0,0,0,0,1.503338,0,0
```

TFIDF_NONSPARSE/test.arff (Fig. 10)

➤ TFIDF y SPARSE

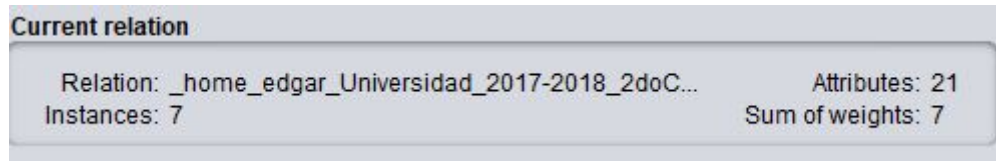
```
java -jar makeCompatible.jar TFIDF_SPARSE/config.txt TFIDF_SPARSE/diccionario test.arff
TFIDF_SPARSE/test.arff
```

```
@attribute no numeric

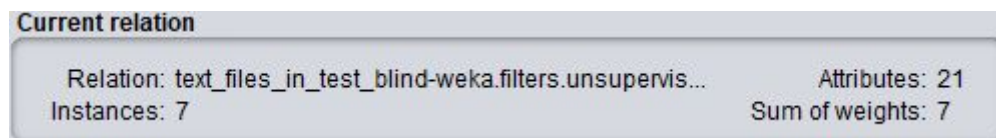
@data
{0 ?,8 1.503338,10 1.503338}
{0 ?,2 2.126041}
{0 ?,12 2.126041}
{0 ?,2 0.867953,3 0.867953,16 0.867953,17 0.867953,19 0.867953,20 0.867953}
{0 ?,2 1.063021,5 1.063021,7 1.063021,13 1.063021}
{0 ?,15 2.126041}
{0 ?,10 1.503338,18 1.503338}
```

TFIDF_SPARSE/test.arff (Fig. 11)

- El último paso será comprobar que efectivamente, por ejemplo, los ficheros 'BOW_SPARSE/train.arff' y 'BOW_SPARSE/test.arff' son compatibles. Los abrimos en Weka (Fig. 12 y 13), y efectivamente son compatibles y tienen la misma cantidad de atributos.



BOW_SPARSE/train.arff (Fig. 12)



BOW_SPARSE/test.arff (Fig. 13)

Detalles relevantes del diseño y la implementación del software

Introducción experimental

Para abordar el problema primeramente implementamos un modelo de compatibilización manual mediante estructuras 'hashmap' para entender a grandes rasgos el funcionamiento del filtro 'StringToWordVector'. Estas estructuras de datos son empleadas para aplicar la compatibilización de ambos conjuntos de instancias train y test, a efectos prácticos será restringir el conjunto de atributos de test a train.

El filtro StringToWordVector(STWV) convierte atributos String (textos) en conjuntos de atributos numéricos (palabras), en función de su configuración obtendremos instancias cuyos atributos representan : pertenencia , frecuencia de aparición o relevancia de palabra con relación a la clase.

➤ Ejemplo:

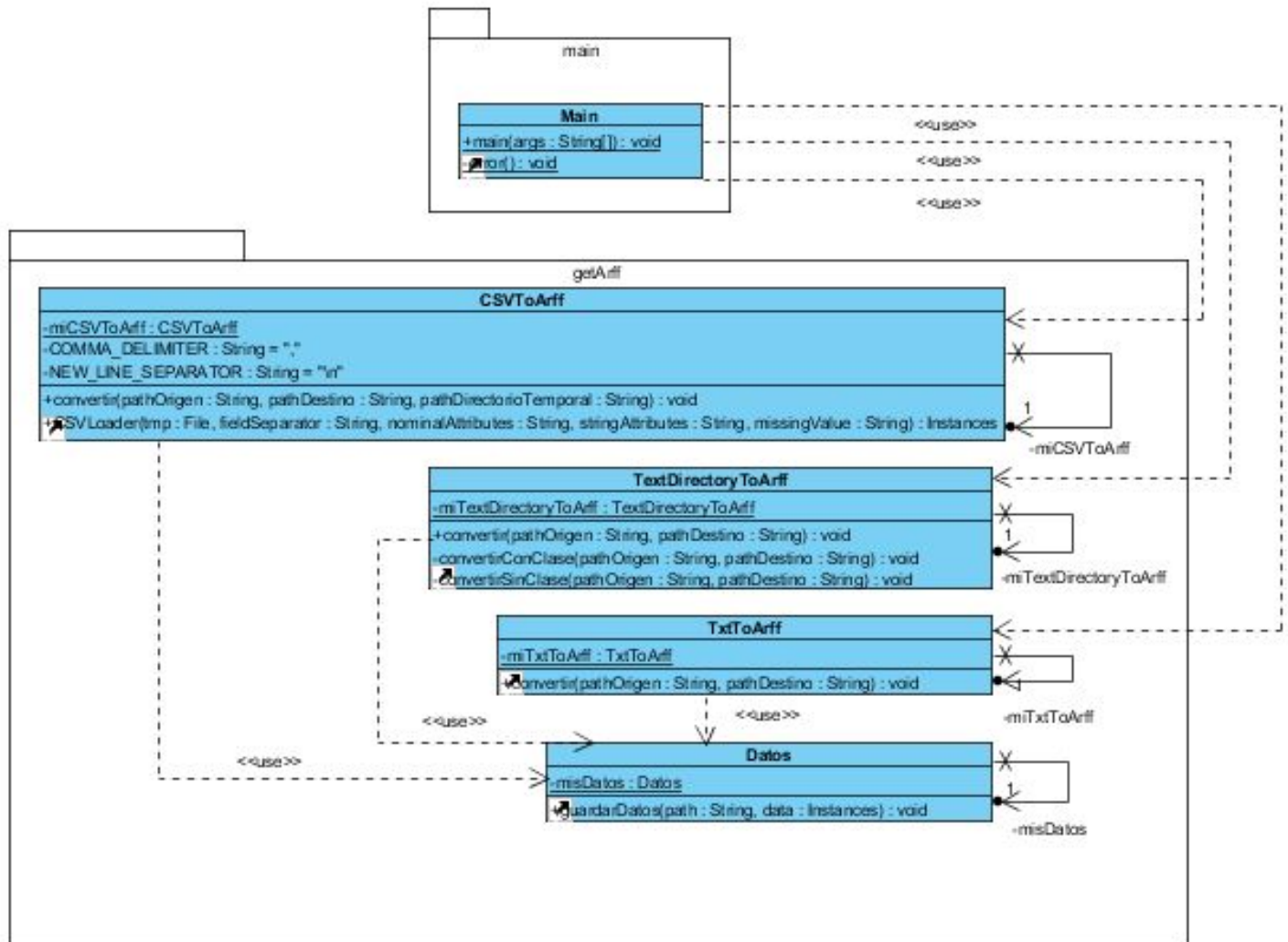
| train | | test | |
|-----------|------------|------------|-----------|
| key | value | key | value |
| 'palabra' | 'posición' | 'posición' | 'palabra' |
| 'a' | 1 | 2 | 'hola' |
| | | | |

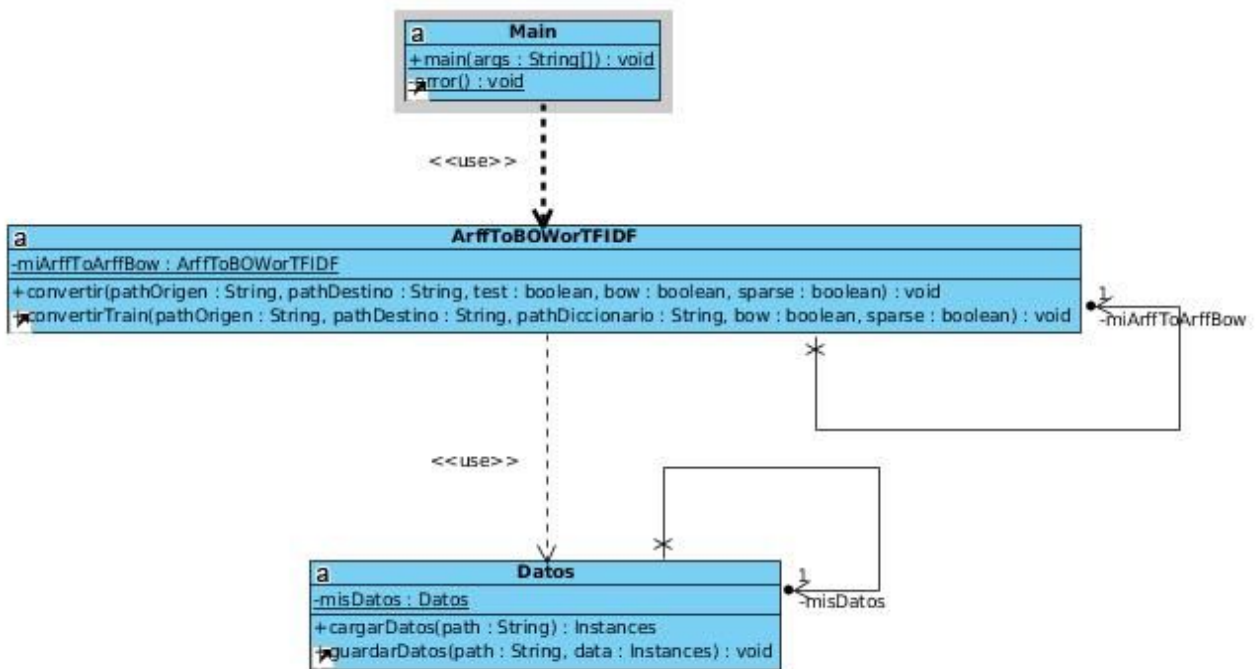
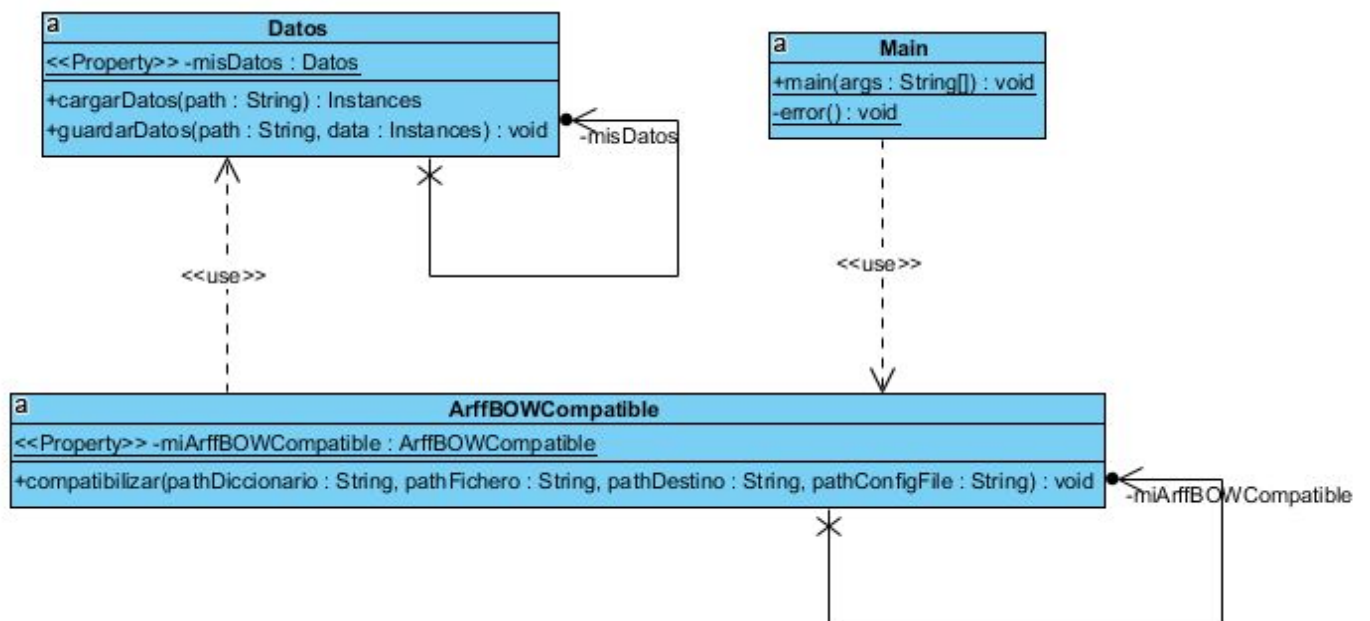
```
@attributes
0 * @ clase {ham}
1 * @ 'a'
2 * @ 'hola'
3 * @ vendiendo
*
* @data
{"hola",ham}          --filtrar--> {0 ham, 2 1}          || {ham,0,1}          //pertenencia
{"a a a",spam}        --filtrar--> {0 spam, 2 3}          || {ham,0,3}
//frecuencia
{"a vendiendo",spam}  --filtrar--> {0 spam, 2 0.05 , 3 1.5 } || {ham, 0.05 , 1.5} //frecuencia
                        * {matriz dispersa}          || {matriz no dispersa}
```

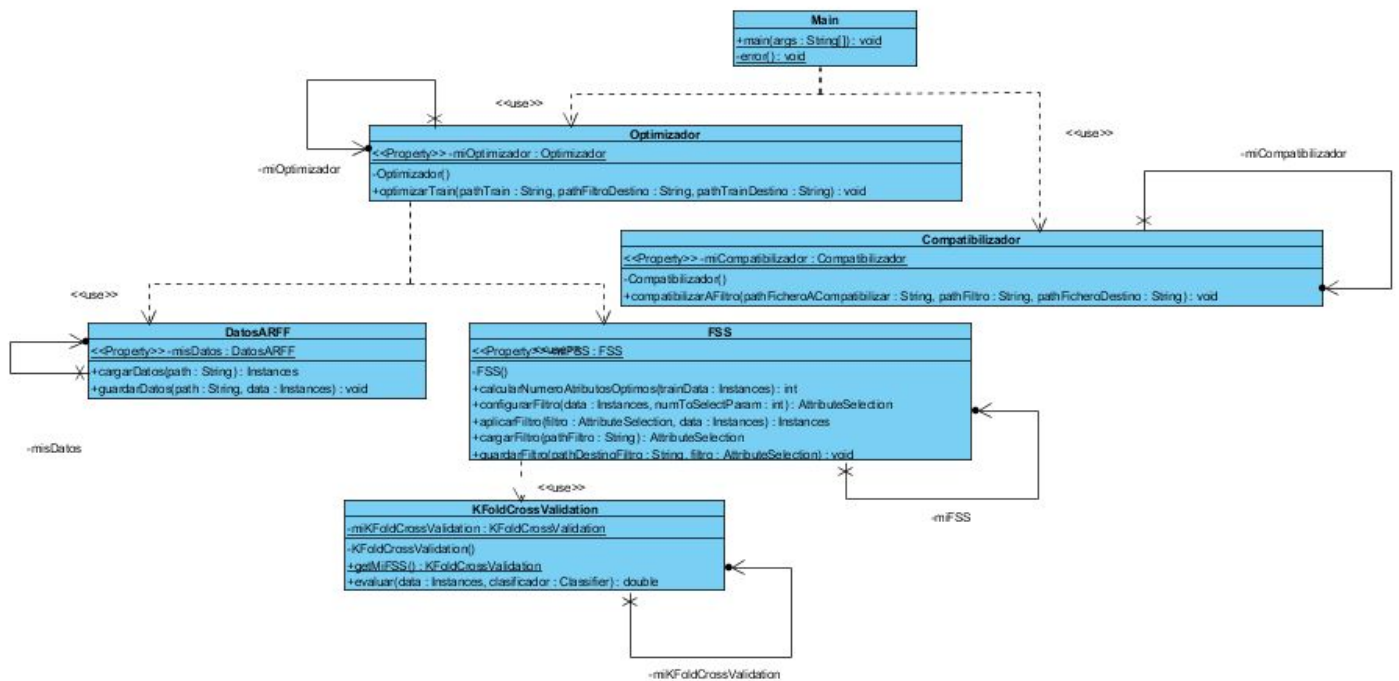
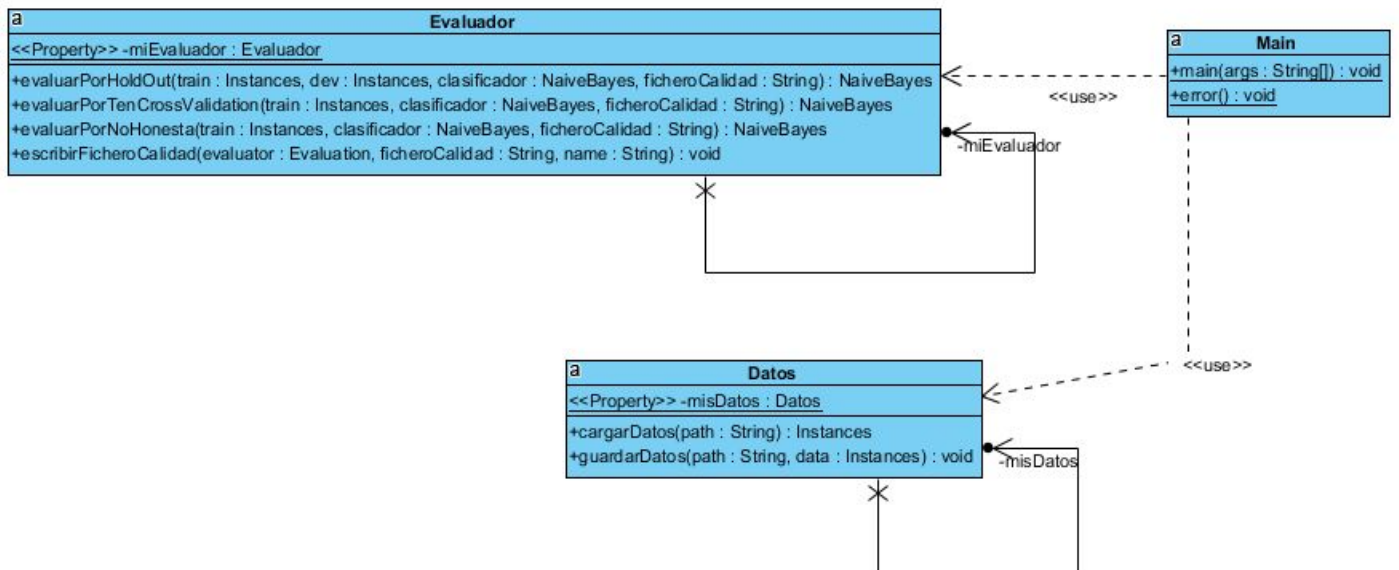
Diseño

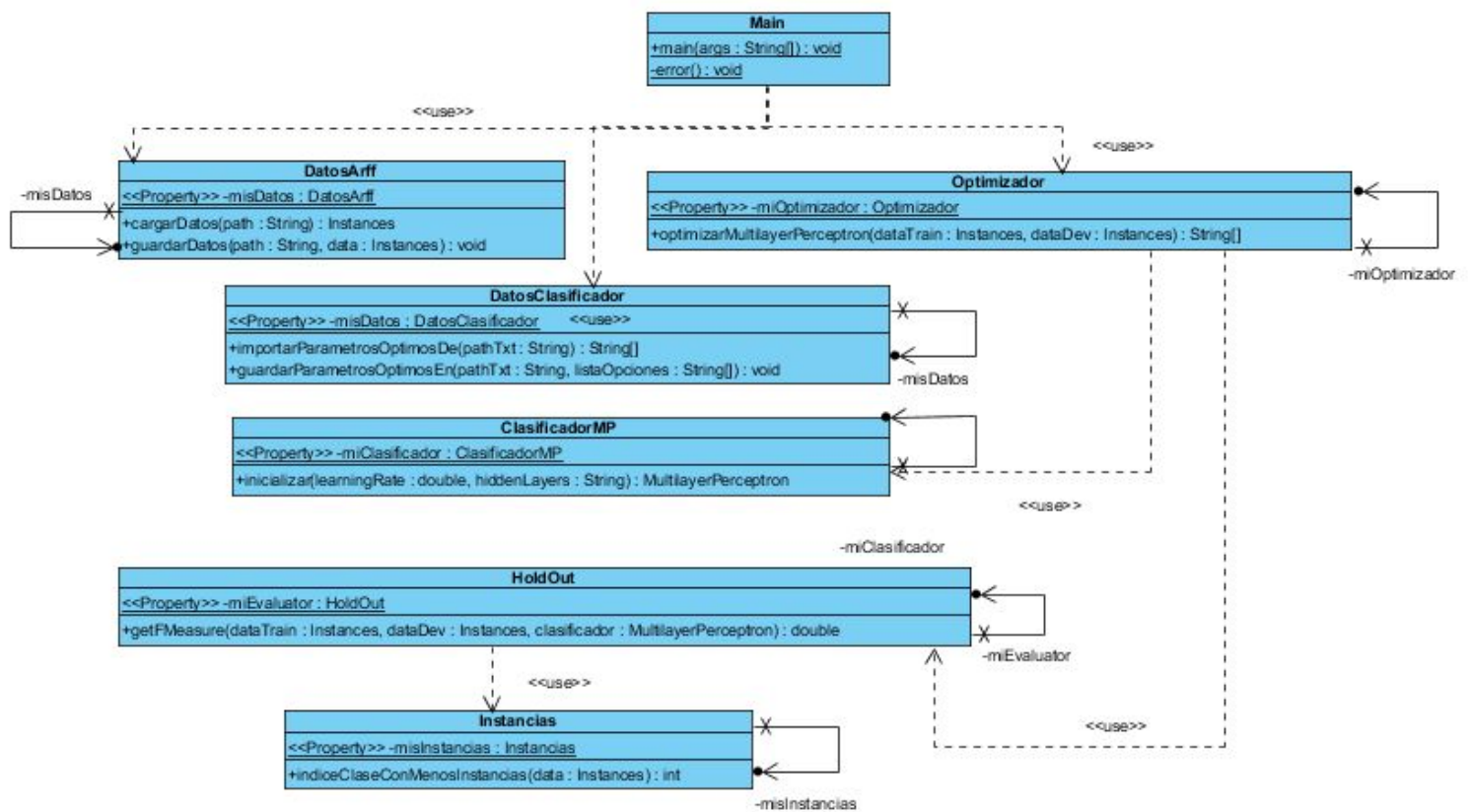
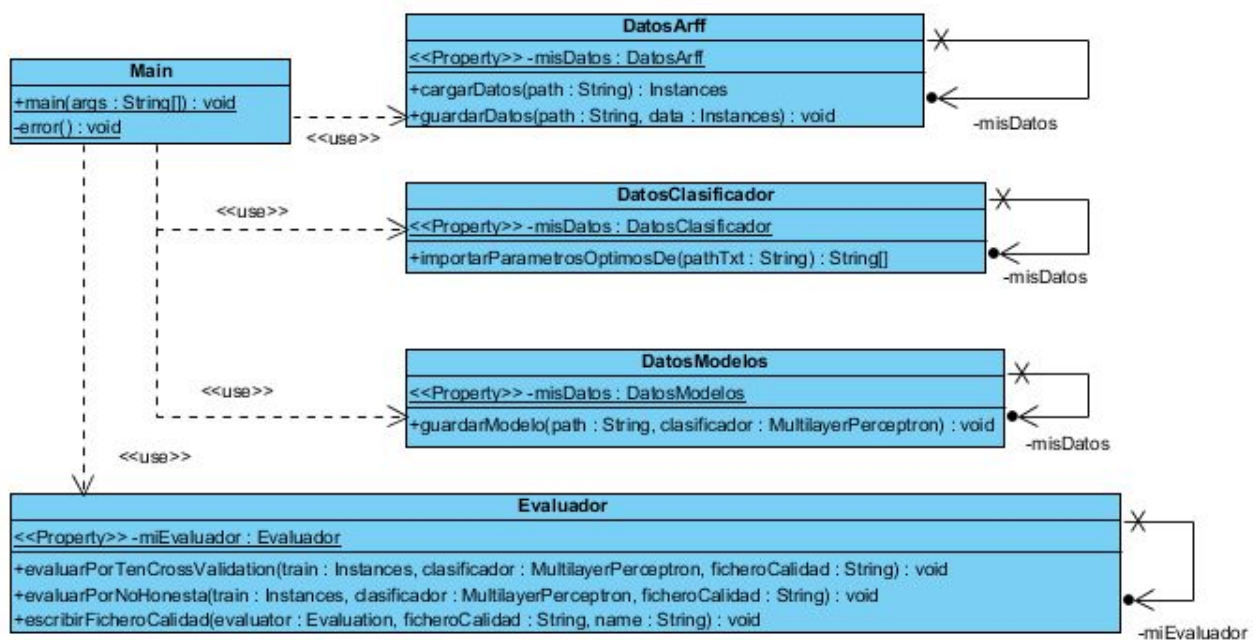
Los diagramas de clases asociados a la aplicación es el siguiente:

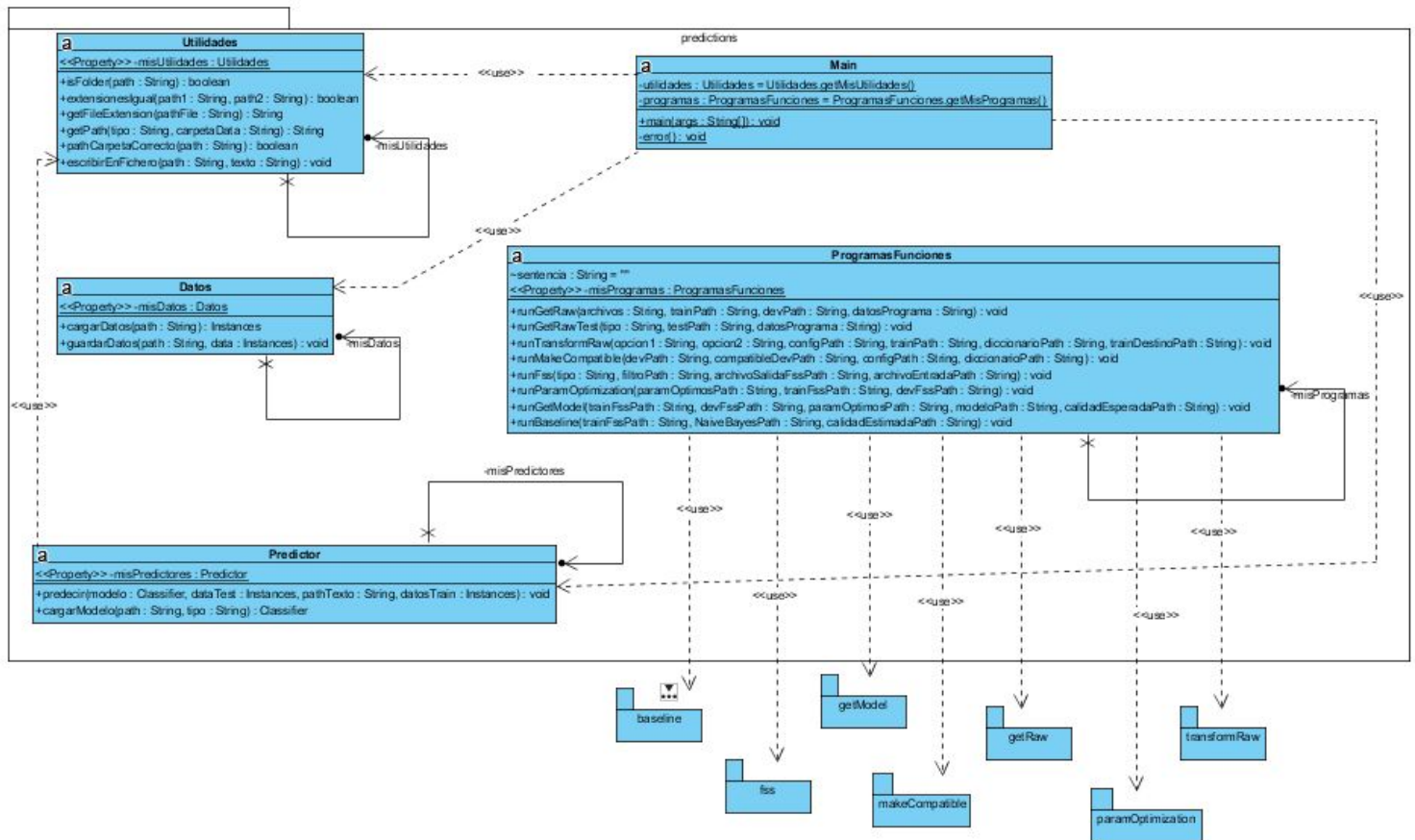
➤ **getRow:**



➤ *transformRaw:*➤ *makeCompatible:*

➤ *fss:*➤ *getBaselineModel:*

➤ **paramOptimization:**➤ **getModel:**

➤ **predictions:****Implementación**

Hemos utilizado la “Librería de Weka” para lograr una implementación más eficiente de nuestro código y reutilizar todas las características facilitadas por sus desarrolladores, las más importantes :

- Uso del método *inputFormat(·)* para la compatibilización de ficheros, al determinar un formato de entrada a un filtro y después aplicarlo a varios conjuntos de instancias logramos que todos tengan el mismo espacio de atributos, el problema es que para que funcione se deben filtrar todos los conjuntos de instancias con el mismo filtro, esta solución no permitía la configurabilidad del filtro.
- Uso de diccionarios para lograr espacios de atributos compatibles entre ficheros ‘train.arff’, ‘dev.arff’ y ‘test.arff’.

Filtro de Weka para filtrar Strings (textos) *StringTowardVector*:

- Una vez aplicado el filtro de Weka, crea un diccionario que se utilizara para compatibilizar el resto de archivos ‘raw.arff’:

- Algoritmos BoW y TFIDF:
 - *setLowerCaseTokens(.)*: configura las instancias para que estén escritas en minúsculas.
 - *setDictionaryFileToSaveTo(.)*: Establecer archivo de diccionario para guardar
- Asigna a los atributos, pesos determinados, por los criterios de los algoritmos utilizados, para seleccionarlos según su capacidad para predecir la clase:
 - Algoritmo TFIDF:
 - *setOutputWordCounts(.)*: Establece recuentos de palabras de salida
 - *setNormalizeDocLength(.)*: Establece la longitud de normalización del documento.
 - *setTFTransform(.)*: Establece la transformación 'TFT'.
 - *setIDFTransform(.)*: Establece la transformación 'IDFT'.
 - Algoritmo BoW: es el algoritmo por defecto del filtro 'STWV'.

Filtro de Weka para compatibilizar y filtrar Strings (textos):

FixedDictionaryStringToWordVector:

- Una vez aplicado el filtro de Weka, crea un diccionario que se utilizara para compatibilizar el resto de archivos '*raw.arff*':

Algoritmos BoW y TFIDF:

- *setOptions(.)*: Establece las opciones.
- *setDictionaryFile(.)*: Establece el diccionario a utilizar.

Hemos podido comprobar que cuando tenemos un atributo nominal [*pos,neg*] uno de los valores no nos lo escribe en los fichero '*.arff*' (Fig.14) en la representación *sparse*, pero hemos comprobado que el programa Weka los carga correctamente. Para realizar esta última comprobación, utilizamos el siguiente fragmento de código (Fig 15), donde listamos los valores nominales de la clase [*pos,neg*] y cuantas instancias hay de cada tipo.

```

1  @attribute {pos,neg} nominal
2  @attribute {0,1} numeric
3  @attribute no numeric
4
5  @data
6  {1 0.877499,8 0.877499,9 0.877499,10 0.593491,14 1.363014}
7  {0 neg,9 1.080705,10 0.730927,18 1.678653}
8  {1 0.651446,4 1.011888,6 1.011888,11 1.011888,12 1.011888}
9  {0 neg,2 0.664447,3 0.664447,16 1.032082,17 1.032082,19 1.032082,20
10 {2 0.740724,5 1.150562,7 1.150562,13 1.150562}
11 {3 1.150856,15 1.787619}
12 {0 neg,8 1.35622,10 0.917271,20 1.35622}
13

```

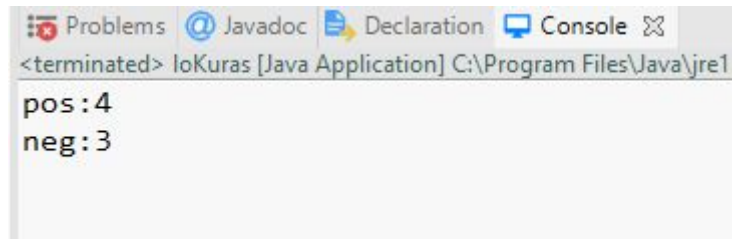
Los valores (*pos*) no se reflejan en el *.arff* (Fig. 14)

Código para realizar la visualización anteriormente mencionada:

```
for(int i=0;i<bowData.numClasses();i++) {  
    System.out.print(bowData.classAttribute().value(i));  
    System.out.println(bowData.attributeStats(bowData.classIndex()).nominalCounts[i]);  
}
```

Código (Fig. 15)

Resultado:



Resultados Código (Fig. 16)

Distribución de tareas

- Documentación:
 - Edgar Andrés Santamaría
 - Paula de Jaime de Toro
 - Maitane Ruiz Monroy
- Diseño:
 - Edgar Andrés Santamaría
 - Paula de Jaime de Toro
 - Maitane Ruiz Monroy
- Implementación:
 - Edgar Andrés Santamaría
 - Paula de Jaime de Toro
 - Maitane Ruiz Monroy
- Pruebas:
 - Edgar Andrés Santamaría
 - Paula de Jaime de Toro
 - Maitane Ruiz Monroy

Selección de atributos²

La selección de atributos hace referencia a un conjunto de técnicas cuyo objetivo es realizar una criba del conjunto de atributos. Estas técnicas están pensadas para lograr eliminar atributos redundantes, así como para eliminar aquellos cuya correlación con la clase es mínima.

Por ello, el uso de herramientas de selección de atributos, como puede ser “Feature Subset Selection” ayuda a reducir el tiempo que se requiere en la clasificación de las instancias.

² Bibliografía: [\[Enunciado - Proyecto: Text Mining - Parte 2: Inferencia del clasificador\]](#)

Clasificación

Baseline³

Cada problema de clasificación es único, y es muy probable que no exista una guía que indique qué atributos descartar y qué algoritmo utilizar. Lo lógico es utilizar una técnica de selección de atributos, e ir probando la tasa de aciertos obtenida con distintos algoritmos.

El problema surge cuando no se sabe si los resultados obtenidos son óptimos. Es inútil utilizar un algoritmo sofisticado y complicado que consume muchos recursos si con un algoritmo simple y sencillo se obtienen los mismos resultados. Ese umbral que indica la validez de lo obtenido con un algoritmo más complejo es el “Baseline”.

Al final, el baseline actúa de punto de referencia, y sin un punto de referencia no se va a saber si se ha conseguido algún progreso en la evaluación [\[Eileen Chappelle, MPH, Cap 4\]](#). Existen diferentes formas de conseguir un resultado “Baseline”, para algunas personas ésto es un resultado aleatorio, y para otras la predicción más común.

Los problemas pueden ser de varios tipos:

- **Clasificación:** Si es un problema de clasificación, se recomiendan algoritmos como el “ZeroR”, “One Rule” o “Naïve Bayes”. Modelos sencillos y que requieren poco tiempo de cómputo. [\[Enunciado - Proyecto: Text Mining - Parte 2: Inferencia del clasificador\]](#)
- **Regresión:** En los problemas de regresión se pueden utilizar medidas como la media o la mediana.
- **Optimización:** Si el problema es uno de optimización, se puede usar un número fijo de muestras aleatorias.

Los resultados del Baseline serán la cota inferior de porcentaje de acierto para el problema, esto puede indicar que el algoritmo cuenta con un amplio rango de mejora. Si no se puede obtener un resultado mejor con un algoritmo más complejo (Multilayer Perceptron) que el del baseline (NaiveBayes), entonces el algoritmo utilizado para resolver el problema puede no ser el apropiado, en ese caso, es recomendable investigar algoritmos más potentes u optimizar la configuración de los existentes ateniéndose a distintas configuraciones.

³ Bibliografía : [\[How To Get Baseline Results And Why They Matter\]](#)

Algoritmo asignado

Multilayer Perceptron ⁴

Es un algoritmo que busca simular el comportamiento de las neuronas del cerebro humano y mediante el mismo, resolver problemas de decisión.

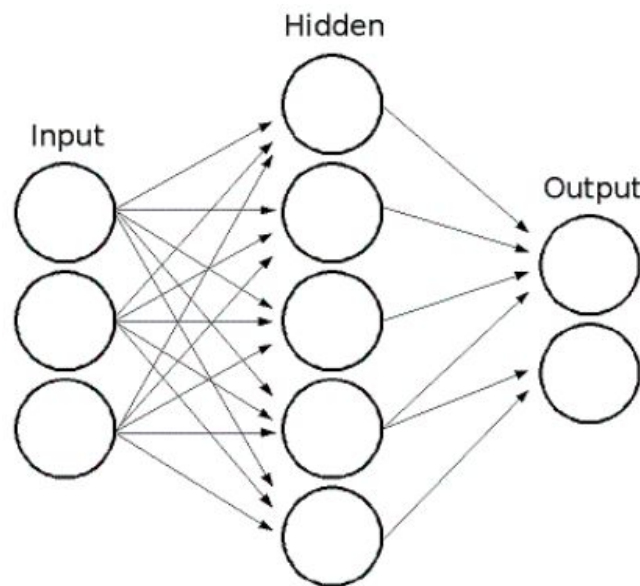
La idea general reside en que el mismo algoritmo, utilizará diferentes representaciones de los datos y les aplicará diferentes funciones compatibles, con el fin de entender los patrones inmersos en los mismos. Para facilitar el uso de estas técnicas weka aporta una implementación en JAVA de manera gratuita. [\[Alpaydin, 2010, Chap. 11\]](#).

La neurona es una unidad de procesamiento básica de este sistema. Las neuronas están interconectadas entre sí y cuentan con una función asociada que se le aplica a las entradas para obtener una salida. Las entradas a una neurona pueden ser salidas de otras o estímulos externos. [\[Alpaydin, 2010, Chap. 11, 2\]](#)

Toda esta red de neuronas está compuesta por tres tipos de capas (*Fig. 17*):

- Capa de entrada (*input layers*): Esta capa no contiene ningún tipo de procesamiento, ya que está compuesta por tantas neuronas como entradas tenga el problema. Es esencial que una red contenga neuronas, en la capa de entrada.
- Capa oculta (*hidden layers*): La red puede tener una o varias capas ocultas, son capas que contienen lógica, y que tienen como fin, llegar a la capa de salida, donde se establece la clasificación.
- Capa de salida (*output layers*): Si la clase es nominal, el número de neuronas será igual al número de valores que puede tomar la clase. En cambio, si la clase es numérica sólo dispondrá de una neurona.

⁴ Bibliografía: [\[Alpaydin, 2010\]](#)



Esquema de capas de la red (Fig. 17)

Las redes neuronales constituyen un sistema universal de aproximación, distinguiéndose el caso trivial (sin capas ocultas) como un estimador lineal y los más complejos como estimadores que utilizan fronteras de decisión. Dichas fronteras se generan mediante las capas ocultas establecidas.

Cualquier tipo de problema es abordable mediante redes neuronales, pero cabe destacar que es un método costoso en cuanto a tiempo, y además difícil de optimizar.

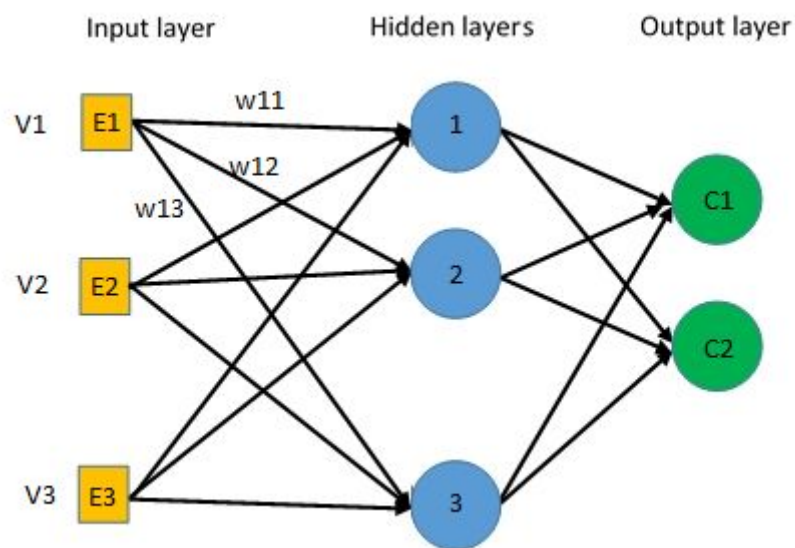
Las interconexiones entre neuronas, además de trasladar valores, cuentan con un peso asociado para diferenciar la preferencia entre las entradas. Estos pesos se modifican continuamente, concretamente al final de cada predicción gracias al algoritmo de "BackPropagation", que consiste en propagar los errores producidos de manera inversa por la red, ajustando los pesos de las aristas por las que pasa. [\[Alpaydin, 2010, Chap. 11.7\]](#)

La función asociada, es de tipo lineal y se aplica a todos los valores de entrada. Tras la aplicación de dicha función, se contrastará con el valor "*bias*" asociado al nodo y se logrará un valor a la salida dependiendo de si supera al bias o no (0, 1), podemos asemejar la función de cada neurona a una función sigmoide.

El algoritmo propuesto por weka cuenta con varios parámetros de optimización entre los cuales destacamos "hiddenLayers" y "training rate", el primero determina el número de capas ocultas que posee la red neuronal, y el segundo la velocidad con que convergerá el problema de entrenamiento. Otros atributos considerados para la optimización son el "momentum", y la tasa de entrenamiento. El primero actúa de manera similar que el training rate y el segundo define el número de veces que se itera (epoch) el

entrenamiento de la red neuronal, lo que supone un gran coste temporal para el entrenamiento.

- **Ejemplo Práctico**



Esquema sencillo del MultilayerPerceptron (Fig. 18)

Los valores de entrada (V1, V2 y V3) se recibirán en la capa de entrada también llamada “Input Layer”. Cada neurona se relaciona con cada neurona de la siguiente capa, y cada relación tendrá un peso (o “weight”) diferenciado: w11, w12, w13,...

Por ejemplo, la neurona de nombre “1” decidirá si esta se activa basándose en el sumatorio de los pesos de sus entradas.

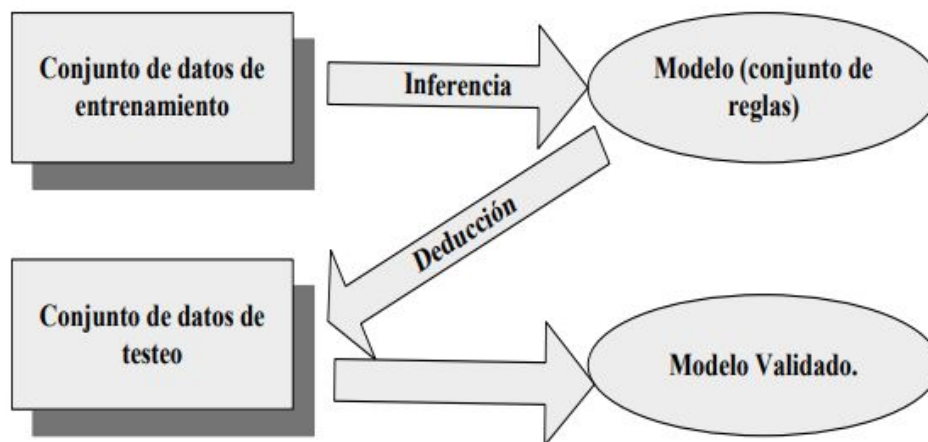
El sumatorio será la entrada recibida en E1 por w11, en E2 por w21 y en E3 por w31

$$\begin{aligned} \sum (\text{pesos de entradas en la Neurona 1}) &= \\ &= [V(E1) * w11] + [V(E2) * w21] + [V(E3) * w31] \end{aligned}$$

El valor obtenido del sumatorio se evaluará mediante una función que determinará si la neurona debe ser activada o no.

Diferencia entre la fase de inferencia del modelo y la fase de aplicación del modelo para hacer las predicciones⁵

El proceso que se utiliza para generar un modelo validado es el siguiente:



Esquema de la generación de un modelo (Fig. 19)

En esta fase, se refleja el uso de la inferencia para obtener el modelo de nuestro conjunto de datos (train.arff), donde es recogido el conjunto de reglas que deberán cumplir las instancia a testear.

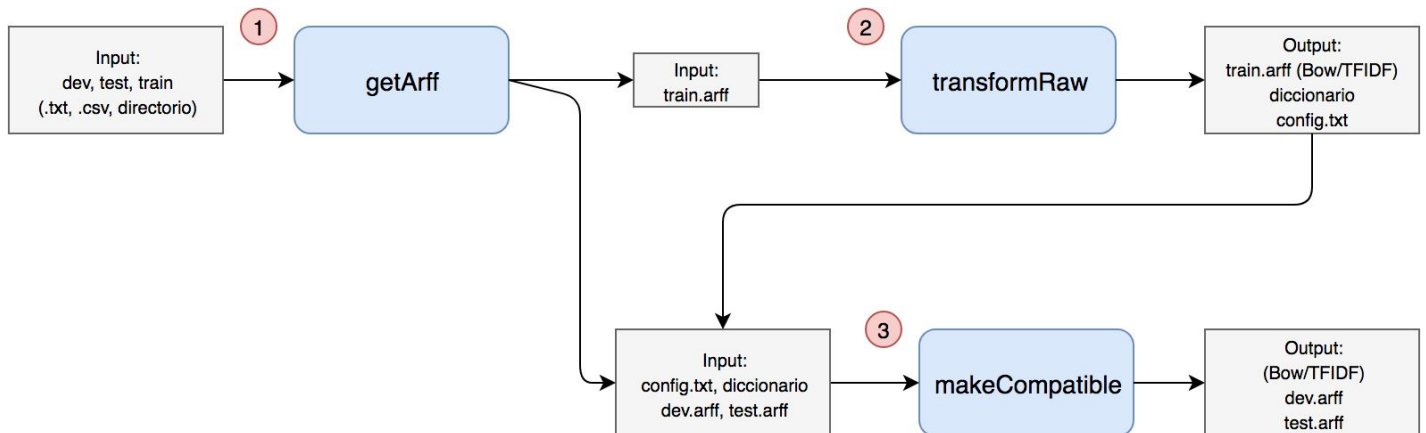
La aplicación del modelo para realizar las predicciones, es una fase que se realiza con un modelo ya creado con anterioridad. En ella, se realiza una clasificación de las instancias a testear, mediante el uso del modelo, anteriormente nombrado.

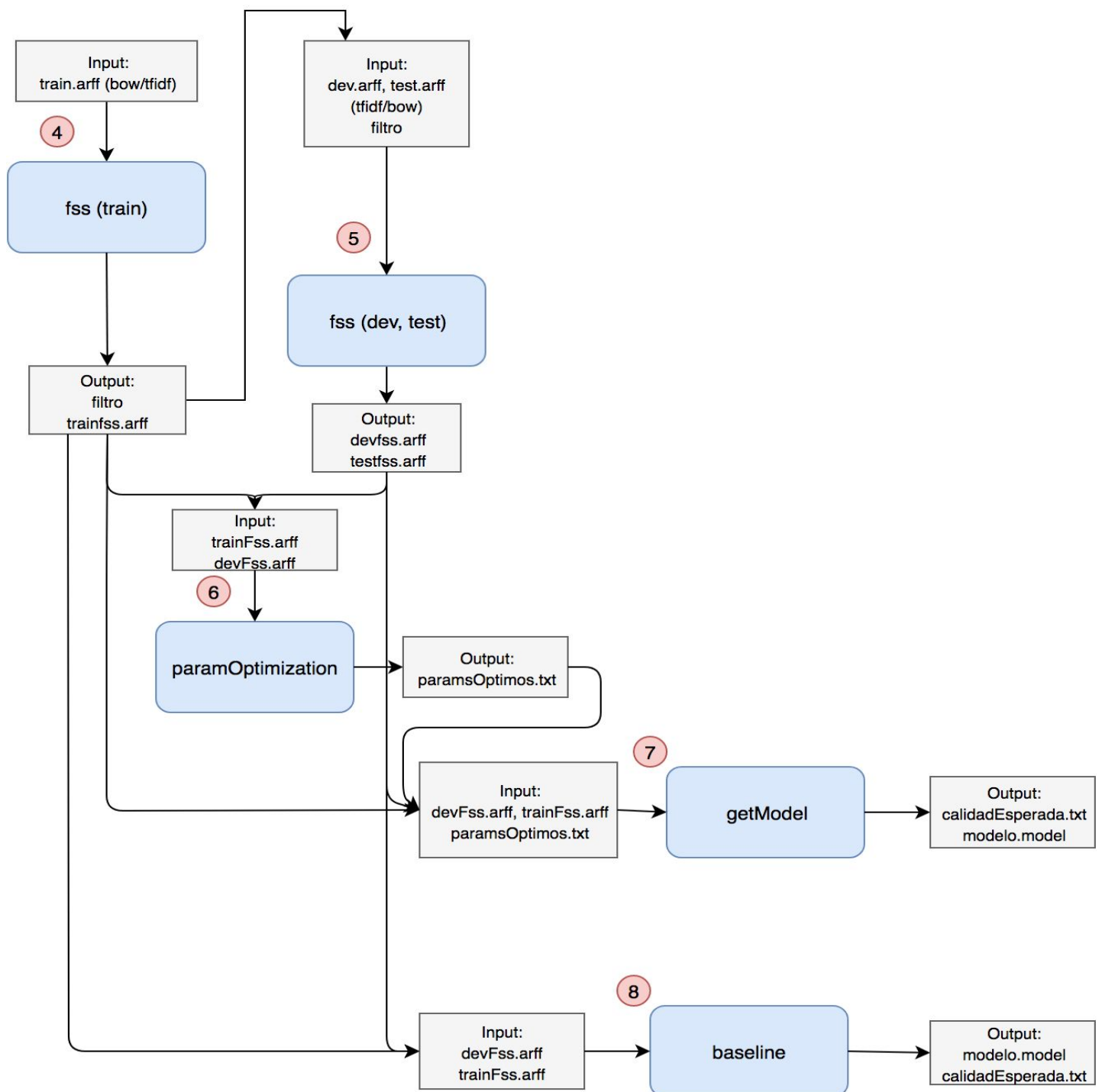
⁵ Bibliografía: [\[Aplicación de algoritmos de clasificación supervisada usando Weka.\]](#)

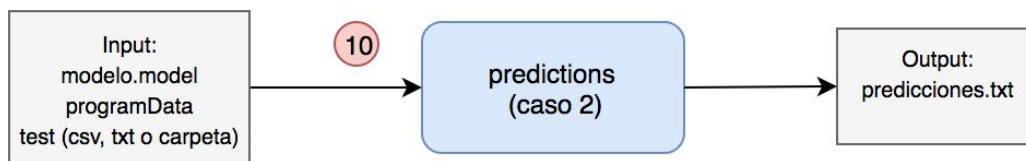
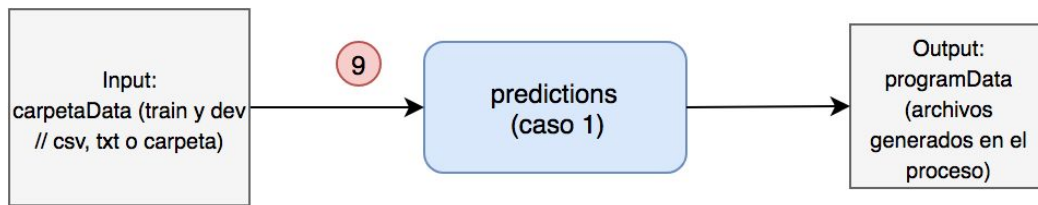
Marco experimental

Gráfico del funcionamiento de los software

Adquisición y pre-proceso de datos



Inferencia del clasificador

Predicciones

Pruebas realizadas

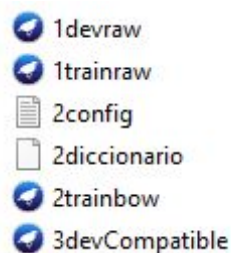
Adquisición y pre-proceso de datos

```
Se esta ejecutando el programa "getArff"  
convertir texto a .arff....  
convertir texto a .arff....  
El programa "getArff" ha terminado.  
Se esta ejecutando el programa "transformRaw"  
El programa "transformRaw" ha terminado.  
Se esta ejecutando el programa "makeCompatible"  
El programa "makeCompatible" ha terminado.
```

Ejecución de los programas *getArff*, *transformRaw*, *makeCompatible* (Fig. 20)

Como hemos comentado anteriormente, este paquete de software está compuesto por los módulos “*getArff*”, “*transformRaw*” y “*makeCompatible*”.

Dichos módulos se han probado con *movies_reviews*, *tweets*, y *sms_spam*, y la salida ha sido la siguiente:



1devraw
1trainraw
2config
2diccionario
2trainbow
3devCompatible

Archivos obtenidos de la ejecución de *getArff*, *transformRaw*, *makeCompatible* (Fig. 21)

- “*1trainraw.arff*” y “*1devraw.arff*” son compatibles.
- “*2config*”.txt establece si es sparse/non-sparse y bow/tfidf.
- “*2trainbow.arff*” será train después de aplicar el filtro StringToWordVector.
- “*2diccionario*” guarda el conjunto de atributos (el vocabulario) usado en los ficheros bow/tfidf.
- “*3devCompatible.arff*” será compatible con “*2trainbow.arff*”.

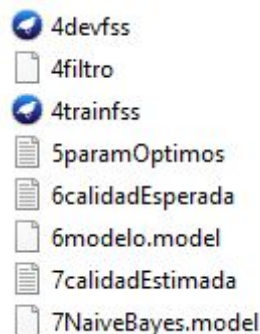
Inferencia del clasificador

```
Se esta ejecutando el programa "fss"
El programa "fss" ha terminado.
Se esta ejecutando el programa "fss"
El programa "fss" ha terminado.
Se esta ejecutando el programa "paramOptimization"
40/40
El programa "paramOptimization" ha terminado.
Se esta ejecutando el programa "getModel"
El programa "getModel" ha terminado.
Se esta ejecutando el programa "baseline"
El programa "baseline" ha terminado.
```

Ejecución de los programa fss, paramOptimization y getModel (Fig. 23)

Como hemos comentado anteriormente, este paquete de software está compuesto por los módulos “fss”, “paramOptimization”, “baseline” y “getModel”.

Dichos módulos se han probado con *movies_reviews* y la salida ha sido la siguiente:



Archivos obtenidos de la ejecución de fss, paramOptimization y getModel (Fig. 22)

- “4trainfss.arff” y “4devfss.arff” serán los ficheros resultantes de aplicar el filtro “Feature Subset Selection”. Son completamente compatibles.
- “4filtro” se usa para compatibilizar el fichero dev con “4trainfss.arff”
- “5paramOptimos.txt” guarda los parámetros óptimos del ‘MultilayerPerceptron’
- “6modelo.model” será el clasificador ‘MultilayerPerceptron’ ya configurado óptimamente.
- “6calidadEsperada” guardará la evaluación del clasificador anterior.
- “7NaiveBayes.model” será el clasificador ‘Baseline’.
- “7calidadEstimada” guardará la evaluación del ‘Baseline’.

Predicciones

➤ Caso 1: Generar modelo.

En este caso, se realiza el procedimiento de los dos anteriores casos (adquisición y pre-proceso de datos y inferencia del clasificador) , y generando los mismos archivos.

Se ejecutan consecutivamente un programa detrás de otro, sirviéndose de los archivos generados por los anteriores programas ejecutados.

➤ Caso 2 : Predecir las clases de test.

- Baseline

```
C:\Users\Paula\Google Drive\UPV_EHU\3º\2º Cuatrimestre\SAD\Proyecto\parte 3\pruebas>java -jar predictions3.jar -p -b "movies_reviews_data\7NaiveBayes.model" "movies_reviews_data" "movies_reviews\test_blind"
=== CASO 2 : Predecir las clases de test ===
Se esta ejecutando el programa "getArff"
convertir directorio a .arff...
El programa "getArff" ha terminado.
Se esta ejecutando el programa "makeCompatible"
El programa "makeCompatible" ha terminado.
Se esta ejecutando el programa "fss"
El programa "fss" ha terminado.
Cargando datos de movies_reviews_data/4trainfss.arff...
Cargando datos de movies_reviews_data/4devfss.arff...
Cargando datos de movies_reviews_data/4testfss.arff...
Se esta ejecutando la predicción de la clase
Escribiendo fichero...
La predicción de la clase ha terminado.
```

Ejecución de los predictions mediante el modelo Baseline (Fig. 24)

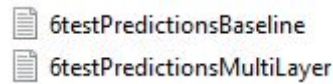
- Multilayer Perceptron

```
C:\Users\Paula\Google Drive\UPV_EHU\3º\2º Cuatrimestre\SAD\Proyecto\parte 3\pruebas>java -jar predictions3.jar -p -m "movies_reviews_data\6modelo.model" "movies_reviews_data" "movies_reviews\test_blind"
=== CASO 2 : Predecir las clases de test ===
Se esta ejecutando el programa "getArff"
convertir directorio a .arff...
El programa "getArff" ha terminado.
Se esta ejecutando el programa "makeCompatible"
El programa "makeCompatible" ha terminado.
Se esta ejecutando el programa "fss"
El programa "fss" ha terminado.
Cargando datos de movies_reviews_data/4trainfss.arff...
Cargando datos de movies_reviews_data/4devfss.arff...
Cargando datos de movies_reviews_data/4testfss.arff...
Se esta ejecutando la predicción de la clase
Escribiendo fichero...
La predicción de la clase ha terminado.
```

Ejecución de los predictions mediante el modelo MultilayerPerceptron (Fig. 25)

Se realiza una predicción del archivo test mediante el clasificador, prediciendo las clases de las instancias de dicho archivo.

Los ficheros generados se pueden ver en la Figura 26.



Archivos obtenidos de la ejecución de `predictions` (Fig. 26)

- `6testPredictionsBaseline.txt` es la comparación de la clase real con la clase predicha por el *Baseline*.

```

1  ===== Predictions on test set =====
2
3      inst#      actual  predicted  error  prediction
4          1        1:~    1:neg      0.996
5          2        1:~    1:neg      0.819
6          3        1:~    1:neg      1
7          4        1:~    2:pos      1
8          5        1:~    2:pos      0.991
9          6        1:~    2:pos      0.993
10         7        1:~    2:pos      1
11         8        1:~    2:pos      1
12         9        1:~    2:pos      0.803
13        10        1:~    1:neg      1
14        11        1:~    2:pos      0.997

```

Archivos obtenidos de la ejecución de `predictions` (Fig. 27)

- `6testPredictionsMultilayer.txt` es la comparación de la clase real con la clase predicha por el *MultilayerPerceptron*.

```

1  ===== Predictions on test set =====
2
3      inst#      actual  predicted  error  prediction
4          1        1:~    1:neg      0.94
5          2        1:~    1:neg      0.94
6          3        1:~    1:neg      0.94
7          4        1:~    2:pos      0.957
8          5        1:~    2:pos      0.957
9          6        1:~    2:pos      0.957
10         7        1:~    2:pos      0.957
11         8        1:~    2:pos      0.957
12         9        1:~    1:neg      0.939
13        10        1:~    1:neg      0.94
14        11        1:~    2:pos      0.957
15        12        1:~    2:pos      0.957

```

Archivos obtenidos de la ejecución de `predictions` (Fig. 28)

Enlace resultados (movies):

<https://drive.google.com/drive/folders/1QtB8IM4qc0ZgHwAewKzIBvZhAeipc3TJ?usp=sharing>

Resultados experimentales

Son dos los marcos experimentales llevados a cabo, que son los siguientes:

Se comparan los ficheros BOW y TFIDF con sus respectivos tras aplicar el filtrado de Meta-Classfier (Fss-InfoGain).

Los ficheros correspondientes se pueden encontrar en el siguiente enlace:

Enlace: <https://drive.google.com/drive/folders/1DhCsikP85HM95yrCrdZ3zTOsiquo1HMc?usp=sharing>

➤ Data (BoW y Non-Sparse):

| | Pre-processed data | | |
|---------------------|--------------------|------|------|
| | Train | Dev | Test |
| Nominal Atributes | 1 | 1 | 1 |
| Numeric Atributes | 1201 | 1201 | 1201 |
| String Atributes | 0 | 0 | 0 |
| Bolean Atributes | 0 | 0 | 0 |
| Atributes total | 1202 | 1202 | 1202 |
| Instances \oplus | 590 | 216 | - |
| Instances \ominus | 610 | 184 | - |
| Instances total | 1200 | 400 | 400 |

Tabla 1.- Datos del fichero arff BOW y Non-Sparse

➤ Data (BoW y Non-Sparse) tras el filtrado de Meta-Classfier (Fss-InfoGain)):

| | Pre-processed data | | |
|---------------------|--------------------|-----|------|
| | Train | Dev | Test |
| Nominal Atributes | 1 | 1 | 1 |
| Numeric Atributes | 251 | 251 | 251 |
| String Atributes | 0 | 0 | 0 |
| Bolean Atributes | 0 | 0 | 0 |
| Atributes total | 252 | 252 | 252 |
| Instances \oplus | 590 | 216 | - |
| Instances \ominus | 610 | 184 | - |
| Instances total | 1200 | 400 | 400 |

Tabla 2.- Datos del fichero arff BOW y Non-Sparse tras selección de atributos

➤ **Data (TFIDF y Non-Sparse):**

| | Pre-processed data | | |
|---------------------|--------------------|------|------|
| | Train | Dev | Test |
| Nominal Atributes | 1 | 1 | 1 |
| Numeric Atributes | 1201 | 1201 | 1201 |
| String Atributes | 0 | 0 | 0 |
| Bolean Atributes | 0 | 0 | 0 |
| Atributes total | 1202 | 1202 | 1202 |
| Instances \oplus | 590 | 216 | - |
| Instances \ominus | 610 | 184 | - |
| Instances total | 1200 | 400 | 400 |

Tabla 3.- Datos del fichero arff TFIDF y Non-Sparse

➤ **Data (TFIDF y Non-Sparse) tras el filtrado de Meta-Classfier (Fss-InfoGain)):**

| | Pre-processed data | | |
|---------------------|--------------------|-----|------|
| | Train | Dev | Test |
| Nominal Atributes | 1 | 1 | 1 |
| Numeric Atributes | 126 | 126 | 126 |
| String Atributes | 0 | 0 | 0 |
| Bolean Atributes | 0 | 0 | 0 |
| Atributes total | 127 | 127 | 127 |
| Instances \oplus | 590 | 216 | - |
| Instances \ominus | 610 | 184 | - |
| Instances total | 1200 | 400 | 400 |

Tabla 4.- Datos del fichero arff TFIDF y Non-Sparse tras selección de atributos

Resultados finales de los experimentos**- BOW y Non-Sparse**

| | Class | Resubstitution error | | | 10-fold Cross Validation | | |
|-----------------|-----------|----------------------|--------|-----------|--------------------------|--------|-----------|
| | | Precision | Recall | F-Measure | Precision | Recall | F-Measure |
| Baseline | \oplus | 0.889 | 0.841 | 0.864 | 0.820 | 0.788 | 0.804 |
| | \ominus | 0.854 | 0.898 | 0.875 | 0.803 | 0.833 | 0.817 |
| | W.Avg. | 0.871 | 0.870 | 0.870 | 0.811 | 0.811 | 0.811 |
| Model | \oplus | 0.951 | 0.943 | 0.947 | 0.856 | 0.834 | 0.845 |
| | \ominus | 0.943 | 0.951 | 0.947 | 0.836 | 0.858 | 0.846 |
| | W.Avg. | 0.947 | 0.947 | 0.947 | 0.846 | 0.846 | 0.846 |

Tabla 5.- Comparación baseline vs model del BOW y Non-Sparse

- TFIDF y Non-Sparse

| | Class | Resubstitution error | | | 10-fold Cross Validation | | |
|-----------------|-----------|----------------------|--------|-----------|--------------------------|--------|-----------|
| | | Precision | Recall | F-Measure | Precision | Recall | F-Measure |
| Baseline | \oplus | 0.855 | 0.817 | 0.835 | 0.771 | 0.742 | 0.756 |
| | \ominus | 0.830 | 0.866 | 0.848 | 0.759 | 0.787 | 0.773 |
| | W.Avg. | 0.842 | 0.842 | 0.842 | 0.765 | 0.765 | 0.765 |
| Model | \oplus | 0.850 | 0.927 | 0.887 | 0.815 | 0.845 | 0.829 |
| | \ominus | 0.918 | 0.834 | 0.874 | 0.836 | 0.805 | 0.820 |
| | W.Avg. | 0.884 | 0.881 | 0.880 | 0.825 | 0.825 | 0.825 |

Tabla 6.- Comparación baseline vs model del TFIDF y Non-Sparse

Espacio de representación

Las cantidades de atributos obtenidas después de aplicar el fss-infogain han sido las siguientes:

- *TFIDF*: 127 atributos.
- *BOW*: 252 atributos.

Aún habiendo menos atributos en *TFIDF* es preferible el modelo *BOW* obtenido. Recordando la definición de Recall “*de los que son positivos cuantos ha identificado el sistema como positivos*”, comparamos la media de los recalls obtenidos mediante el 10-fold-cross-validation en las tablas 5 y 6 (pág 25).

En la *tabla 5* observamos que la media del recall según la evaluación 10-fold-cross-validation es de 0.846 y en la *tabla 6*, en cambio obtenemos un 0.825. Podemos ver que el recall es mayor en la evaluación de los ficheros *BOW*, y esto se aprecia en los porcentajes finales de aciertos.

El porcentaje de aciertos obtenidos mediante 10-fold-cross-validation en **BOW** es de **84.5625%**, y en **TFIDF** es de **82.5%**.

El mayor porcentaje de aciertos que se podría obtener en la evaluaciones anteriores reside en el hold-out. Si el recall es mayor en el hold-out significa que todavía hay margen de mejora.

Aplicación de selección de atributos

La reducción del espacio de representación, es decir, la selección de atributos logra mejores resultados de clasificación tanto en el modelo ‘*BaseLine*’ como en el modelo complejo ‘*MultilayerPerceptron*’. Además, reduce notablemente el tiempo de entrenamiento del clasificador, permitiendo entrenamientos más exhaustivos. Por tanto, determinamos que es conveniente aplicar la selección de atributos. [\[Performing attribute selection\]](#)

Cuando aplicamos el filtro ‘*AttributeSelection*’ junto con ‘*InfoGainAttributeEval*’ y Ranker lo que hacemos es ordenar los atributos según su evaluación individual.

El proceso utilizado es el siguiente:

1. Descartamos aquellos atributos que hayan obtenido un ‘0’ en el ranking.
2. Para los demás atributos obtenemos el porcentaje de aciertos por el método de ‘*10-Fold-Cross-Validation*’. Borrados el atributo cuyo ranking es menor, y volvemos a calcular el porcentaje de aciertos. Si el porcentaje es mayor, se repetirá este paso. En cambio, si éste sigue igual, se recupera el atributo eliminado y se termina.

Comparativa de modelos

Se ha maximizado el “*f-measure*” de la clase minoritaria para la obtención de los parámetros óptimos que forman la configuración del clasificador por el que se obtienen los resultados de las tablas 5 y 6, .

El “recall” obtenido mediante la evaluación 10-fold-cross-validation puede alcanzar una cota superior mayor tanto para el modelo “*MultilayerPerceptron*”, como para el Baseline “*NaiveBayes*”.

En el caso del espacio de representación BoW y Non-Sparse, el “recall” obtenido mediante el método de evaluación anterior es de 84%, y está a 3% de la cota superior del Baseline 87%. Asumimos que a pesar de no haber optimizado plenamente la red neuronal, hemos logrado un resultado parecido a un Baseline optimizado, por tanto, se valora como satisfactoria la optimización del modelo.

Por otro lado, cabe destacar que el modelo se encuentra a un 10% por debajo de su cota superior (94.7%), por lo que merece la pena el uso del algoritmo “*MultilayerPerceptron*”, puesto que podría llegar a ser muy preciso.

Los resultados respecto a TFIDF Non-Sparse son similares, reafirmando lo comentado previamente.

Conclusiones

El objetivo del proyecto es desarrollar una aplicación software capaz de filtrar correo basura, “filtro antispam”, haciendo uso de la minería de palabras o “Text Mining”, que consiste en encontrar patrones en las palabras y, en este caso, clasificar los textos en dos tipos: textos basura (spam) o texto valioso (ham).

El primer paso de la aproximación al software, es preprocesar los formatos habituales en los que se encuentran los datos de textos a clasificar: texto (.txt), archivos separados por comas (.csv) y directorio de textos (text directory). Para esta sección proponemos el módulo **getRaw**, capaz de transformar datos en bruto, “Raw Data”, a archivos con relaciones de atributos (.arff) compatibles con las librerías de Weka implementadas en JAVA, que posteriormente utilizaremos para optimizar y clasificar los datos.

Como segundo paso, proponemos los módulos **transformRaw** y **makeCompatible**. Con ellos convertiremos el espacio de representación de las instancias, hasta ahora tipos “String” que representan los textos, en uno compuesto por tipos “numeric”, que representen las palabras que componen los textos.

Los filtros “*FixedDictionaryStringToWordVector*” y “*StringToWordVector*” referencian a los algoritmos BoW y TFIDF, que son los distintos algoritmos propuestos para la resolución de la tarea.

Para el tercer paso, proponemos los módulos **paramOptimization**, **getBaselineModel**, **getModel** y **fss**. Se entrenará un modelo de inferencia mediante un algoritmo de entrenamiento (*Machine Learning*). El clasificador “*MultilayerPerceptron*” nos proporciona la implementación de una red neuronal para realizar la tarea de clasificación.

Este algoritmo es muy dependiente de la optimización de sus parámetros, por lo que se propone un software de ajuste. Por otro lado, se evalúa el modelo y se contrasta con otro clasificador más simple (*NaiveBayes*) con el fin de tener una base sobre la que comparar resultados. Este proceso es experiencial y busca el continuo contraste de resultados para lograr el conjunto de parámetros más óptimo para los datos propuestos.

El filtro “*FeaturesSubsetSelection*” logra la reducción del espacio de representación sin pérdida de precisión en la tarea de clasificación, en este ámbito decidimos aplicar una selección según el algoritmo de ganancia de información.

El algoritmo Baseline utilizado para la prueba es “*NaiveBayes*” por su eficiencia y rapidez de entrenamiento. Sin optimizar los parámetros, su porcentaje de aciertos es muy próximo a su cota superior obtenida mediante la evaluación “No-Honesta” y supera al “*MultilayerPerceptron*” sin realizar ninguna acción de optimización.

Tras las pruebas realizadas, concluimos que el espacio de representación *BoW* + *fss infoGain* y el algoritmo de aprendizaje “*Multilayer Perceptron*” son una combinación ideal para lograr una gran precisión en la clasificación. Aún así hay que tener en cuenta que éste algoritmo tarda bastante más en construir el modelo, y qué si es rapidez lo que se busca, el clasificador elegido como Baseline podría llegar a ser una buena opción en términos de rapidez y eficacia.

El proyecto consistía en 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, negativo o neutro.

A través de los distintos módulos propuestos se han conseguido ficheros compatibles *.arff* desde datos en bruto, y se han aprendido a utilizar éstos para lograr predecir la clase de cualquier instancia que carezca de clase.

Se ha logrado un porcentaje de aciertos de hasta el **84%** con el algoritmo *MultilayerPerceptron*, y un **81%** con el baseline (*NaiveBayes*).

El código usado en el proyecto cumple sus funciones siempre que los datos de entrada estén entre los siguientes conjuntos de datos: “*movies_reviews*”, “*sms_spam*” y “*tweet_sentiment*”. El módulo encargado de convertir ficheros en “bruto” a ficheros “*.arff*” solo es efectivo y válido para los tipos de datos mencionados anteriormente. Lo ideal sería codificar un conversor universal, es decir, un conversor que no tuviera en cuenta el formato de los datos de entrada.

El rango de valores de los parámetros usados para la optimización del clasificador “*MultilayerPerceptron*” sólo es válido para el conjunto de datos “*movies_reviews*”. Cada conjunto es diferente, y requiere de una configuración diferente, es decir, no se puede asegurar que el mismo rango de valores sea válido para distintos datos.

Si el texto en bruto no está limpio (*caracteres especiales,...*) es poco recomendable seguir adelante con el proceso de predicción. Palabras inexistentes en cualquier idioma, símbolos o números son responsables de entorpecer el proceso de entrenamiento y clasificación del modelo.

Al final, la predicción de la clase se realiza con el modelo entrenado por train (y en ciertas ocasiones con las instancias que componen dev), por ese motivo si los datos de entrenamiento no son óptimos, es muy probable que el resultado tampoco lo sea.

Referencias

- Introducción a la minería de textos:
 - *Minería de textos y análisis de sentimientos en sanidadysalud.com*
[Sergio Rincón García, Máster en Minería de Datos e Inteligencia de Negocios, 2015 - 2016]
http://eprints.ucm.es/39524/1/memoriaTFM_sergio_rincon_garcia.pdf
- Marco teórico. Baseline:
 - [Eileen Chappelle, MPH Pag 4] *Establishing a Baseline as Part of Your Evaluation* (Pàg 4)
https://www.cdc.gov/dhds/pubs/docs/cb_jan2014.pdf
 - *How To Get Baseline Results And Why They Matter*
<https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter>
 - *Enunciado: Proyecto: Text Mining - Parte 2: Inferencia del clasificador*
- Clasificación. Diferencia entre la fase de inferencia del modelo y la fase de aplicación del modelo para hacer las predicciones
 - [Aplicación de algoritmos de clasificación supervisada usando Weka.]
http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/congresos_labsis/cynthia/CNIT_2009_Aplicacion_Algoritmos_Weka.pdf
- Selección de atributos:
 - *Enunciado: Proyecto: Text Mining - Parte 2: Inferencia del clasificador*
- Algoritmo asignado. Multilayer Perceptron:
 - [Alpaydin, 2010]
http://cs.du.edu/~mitchell/mario_books/Introduction_to_Machine_Learning_-_2e_-_Ethem_Alpaydin.pdf
- Marco experimental. Aplicación de selección de atributos:
 - [Performing attribute selection]
<https://weka.wikispaces.com/Performing+attribute+selection>
- Conclusiones:
 - OverView
 - <http://weka.sourceforge.net/doc.stable/>