SMS Spam Filtering

Iván Ridao Freitas
Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil, Buenos Aires, Argentina

1. Introducción

El objetivo de este trabajo es aplicar los conceptos adquiridos concernientes al área de Minería de Datos Web sobre un conjunto de datos y analizar luego los resultados obtenidos. En las secciones siguientes se detalla el trabajo realizado.

2. Dataset

El conjunto de datos seleccionado fue *SMS Spam Collection Data Set*. Esta colección está compuesta por mensajes de texto SMS los cuales han sido etiquetados y recolectados con el fin de investigar el Spam en los teléfonos móviles.

Se llama Spam o correo basura a los mensajes no solicitados, no deseados o de remitente no conocido, habitualmente de tipo publicitario, generalmente enviados en grandes cantidades (incluso masivas) que perjudican de alguna o varias maneras al receptor.

El dataset elegido está compuesto por 5.574 SMS provenientes de distintas fuentes. Toda la información se encuentra en un archivo de texto en el cual cada línea se corresponde con un SMS.

Al principio de cada línea se indica la clase del mensaje y luego el contenido del mismo. Existen dos tipos de clases para los SMS: *spam* y *ham*. La clase *ham* hace referencia a los mensajes que no son Spam. El dataset está formado por 4.827 mensajes *ham* y 747 mensajes *spam*. A continuación se muestra un ejemplo del contenido del dataset:

```
ham     What you doing?how are you?
ham     Ok lar... Joking wif u oni...
ham     U dun say so early hor... U c already then say...
ham     MY NO. IN LUTON 0125698789 RING ME IF UR AROUND! H*
ham     Siva is in hostel aha:-.
ham     Cos i was out shopping wif darren jus now n i called
        him 2 ask wat present he wan lor. Then he started
        guessing who i was wif n he finally guessed darren lor.
spam    FreeMsg Hey there darling it's been 3 week's now and no
        word back! I'd like some fun you up for it still? Tb
        ok! XxX std chgs to send, £1.50 to rcv
spam    URGENT! You have won a 1 week FREE membership in our
        £100,000 Prize Jackpot! Txt the word: CLAIM to No:
        81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
```

Figura 1: Ejemplo del contenido del dataset SMS Spam Collection

Para el posterior pre-procesamiento, clasificación y evaluación de los datos se utilizaron algunos de los algoritmos provistos por la herramienta WEKA. La misma es una plataforma de software para aprendizaje automático y minería de datos escrita en Java y desarrollada en la Universidad de Waikato.

El ingreso de los datos a WEKA se realiza mediante un archivo ARFF (Attribute-Relation File Format), el cual posee un formato específico. Por lo tanto, para poder utilizar la herramienta ha sido

necesario adaptar el dataset a dicho formato. Para ello se codificó una función la cual transforma el dataset en un archivo ARFF válido. Los únicos cambios realizados al contenido de los mensajes fueron relativos a las comillas ya que estas son delimitadoras de strings en ARFF. Por tal motivo se reemplazaron los caracteres " y ' por \" y \' respectivamente. A continuación se muestra un ejemplo del contenido del dataset en formato ARFF:

```
@relation sms_spam

@attribute spam_class { ham, spam }
@attribute text String

@data
ham,   'What you doing?how are you?'
ham,   'Ok lar... Joking wif u oni...'
ham,   'U dun say so early hor... U c already then say...'
ham,   'MY NO. IN LUTON 0125698789 RING ME IF UR AROUND! H*'
ham,   'Siva is in hostel aha:-.'
ham,   'Cos i was out shopping wif darren jus now n i called
him 2 ask wat present he wan lor. Then he started guessing who
i was wif n he finally guessed darren lor.'
spam,  'FreeMsg Hey there darling it\'s been 3 week\'s now and
no word back! I\'d like some fun you up for it still? Tb ok!
XxX std chgs to send, £1.50 to rcv'
spam,  'URGENT! You have won a 1 week FREE membership in our
£100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C
www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18'
```

Figura 2: Ejemplo del contenido del dataset SMS Spam Collection en formato ARFF

3. Clasificación

Como se mencionó anteriormente, se cuenta con un conjunto de datos que incluye distintos textos de ejemplos de SMS y su clase asociada. Utilizando estos datos, se construyeron y se evaluaron diferentes clasificadores de texto con el fin de poder predecir si un cierto mensaje de texto SMS es Spam o no.

3.1. Pre-procesamiento

Para comenzar con el pre-procesamiento de los datos se utilizó una de las principales herramientas para análisis de texto brindada por WEKA: el filtro `StringToWordVector`. De esta forma, los datos resultantes están listos para la etapa de entrenamiento del clasificador.

`StringToWordVector` convierte los atributos de tipo String en un conjunto de atributos representando la ocurrencia de las palabras del texto. En este caso, la conversión se hizo sobre el texto de los SMS. La separación de palabras o tokens se realiza en base a un `tokenizer`, el cual es `WordTokenizer`. Para este último se evaluaron distintos delimitadores ya que una correcta separación de tokens aumenta la precisión del clasificador. Los delimitadores que se utilizaron son:

- *TK-Default*: es el valor por defecto de `WordTokenizer`. Los caracteres a tener en cuenta son " \r\n\t.,;:'"()?!".
- *TK-Complete*: al valor por defecto se agregan diferentes símbolos presentes en el dataset que no se estaban considerando para separar tokens. Los caracteres a tener en cuenta son " \r\n\t.,;:'"()?!-¿¡+*&#$%/=<>[]_`@\\^{}".
- *TK-Numbers*: al delimitador anterior se agregan los números con el fin de eliminar la gran cantidad de tokens que son sólo números y separar aquellos que contienen números. Por ejemplo, los tokens como £10, £1million y £50award serán divididos y se le dará mayor peso a los tokens £, million y award. Los caracteres a tener en cuenta son " \r\n\t.,;:\'\"()?!-¿¡+*&#$%/=<>[]_`@\\^{}|"~0123456789".

Otras opciones que se modificaron en el filtro son: `doNoOperateOnPerClassBasis`, `lowerCaseTokens` y `wordsToKeep`. Las primeras dos son `true`, una porque se quiere recolectar tokens de las distintas clases como un todo y la otra porque el interés principal para este estudio son las palabras, independientemente de si tienen mayúsculas o no. Para `wordsToKeep` se probó con el valor por defecto (1.000) y con un valor muy alto (1.000.000) con el objetivo de incluir todas las palabras del dataset.

Dado que la longitud de los mensajes SMS es corta, la utilización de herramientas de reducción de dimensionalidad tiende a afectar la precisión en el filtrado de Spam. Sólo con el fin de mostrar que este es el caso se realizó una breve evaluación de los resultados de utilizar *stop word removal*, *word stemming* y *attribute selection*. Para el último caso se definió un `MultiFilter` para poder agregar el filtro `AttributeSelection`, cuyo evaluador es `InfoGainAttributeEval` y el buscador es `Ranker` con el `threshold` en cero.

3.2. Algoritmos de clasificación

La selección de los algoritmos de clasificación se realizó teniendo en cuenta cuáles son los más apropiados para la clasificación de texto. Por ello se evaluaron:

- *Support Vector Machines (SVM)*: es probablemente el algoritmo más efectivo para los problemas de clasificación de texto debido a que se enfoca en los ejemplos más relevantes para separar las clases. Se utilizó la clase `SMO` (Sequential Minimum Optimization) de Weka para las pruebas.
- *Naïve Bayes (NB)*: se fundamenta en la teoría de probabilidades, basándose en el teorema de Bayes. Generalmente tiene buenos resultados. Se utilizó la clase `NaiveBayes` de Weka para las pruebas.
- *k-Nearest Neighbors (k-NN)*: ocasionalmente da excelentes resultados en clasificación de texto. Se utilizó la clase `IBk` de Weka para las pruebas, con diferentes valores para `k` (1, 3 y 5).
- *Árboles de Decisión*: es fácil de interpretar y es posible derivar reglas. Se utilizó la clase `PART` de Weka para las pruebas.

Los algoritmos se utilizaron con los valores por defecto para los distintos parámetros de Weka.

También se estudió el método *Boosting*, basado en la construcción de sucesivos clasificadores sobre el conjunto de entrenamiento que se va modificando en función de los errores cometidos por el clasificador anterior. Esta técnica solo se aplicó sobre los algoritmos que mostraron mejores resultados previos ya que el tiempo de entrenamiento y prueba es mucho mayor. El objetivo ha sido mejorar la exactitud de los algoritmos más débiles.

3.3. Pos-procesamiento

Una vez construidos los distintos clasificadores una de las principales tareas que debe realizarse es su evaluación. Para ello se tuvieron en cuenta distintas métricas de efectividad y eficiencia que serán tratadas en detalle en la siguiente sección del informe.

Finalmente, se desarrolló una aplicación en Java la cual permite probar los distintos clasificadores a la hora de filtrar mensajes de texto SMS. Para ello, se filtran los datos del dataset, como se explicó anteriormente, y estos se utilizan para entrenar a los clasificadores. Cada clasificador genera un modelo, el cual se vuelca a un archivo que luego se puede utilizar para predecir si un cierto mensaje de texto SMS es Spam o no. Su desarrollo y modo de utilización se explica en la sección 5.

4. Evaluación

En esta etapa se midió principalmente la efectividad de los clasificadores, es decir la habilidad de tomar decisiones de clasificación correctas. En cuanto a la eficiencia, se detallará el tiempo requerido para las tareas de entrenamiento y prueba.

La evaluación de los filtros, los algoritmos de clasificación y las opciones seleccionadas se realizó utilizando *k-fold cross-validation*. Esta técnica involucra la creación de `k` particiones del conjunto de datos. Para cada uno de los `k` experimentos, `k-1` particiones se usan para entrenamiento y una para prueba. Así, todos los ejemplos son usados para entrenamiento y prueba. El valor seleccionado para `k` es 10, debido a que asegura una evaluación más precisa y además porque es la elección más común para este tipo de técnica.

Una herramienta útil a la hora de calcular los resultados de la evaluación es la matriz de confusión. La misma se define de la siguiente forma:

	Clases predecidas	
Clases verdaderas	*ham*	*spam*
ham	*Verdaderos Positivos (TP)*	*Falsos Positivos (FP)*
spam	*Falsos Negativos (FN)*	*Verdaderos Negativos (TN)*

Figura 3: Matriz de confusión

Las métricas que se evaluaron en los distintos clasificadores son:

- *Falsos Positivos*: El conjunto de mensajes que el clasificador clasificó como pertenecientes a *spam* pero que no pertenecían a tal clase, sino que eran *ham*.
- *Falsos Negativos*: El conjunto de mensajes que el clasificador clasificó como pertenecientes a *ham* pero que no pertenecían a tal clase, sino que eran *spam*.
- *Tasa de Error*: Es el porcentaje de mensajes que se clasificaron incorrectamente. Se calcula de la siguiente forma:

$$tasa\,de\,error = \frac{\#\,de\ \ falsos\ positivos + \#\,de\ \ falsos\,negativos}{\#\,total\,de\,mensajes}$$

- *Exactitud (ACCY)*: Es el porcentaje de mensajes que se clasificaron correctamente. Se calcula de la siguiente forma:

$$exactitud = \frac{\#\,de\,verdaderos positivos + \#\,de\,verdaderos negativos}{\#\,total\,de\,mensajes}$$

- *Recall*: se evalúa por clase y es un valor entre 0 y 1. Su valor aumenta cuando hay pocos falsos negativos. Mide que las instancias de la clase C se clasifiquen como clase C. Se calcula de la siguiente forma:

$$recall(C) = \frac{\#\,de\,verdaderos positivos}{\#\,de\,verdaderos positivos + \#\,de\ \ falsos\,negativos}$$

- *Precisión*: se evalúa por clase y es un valor entre 0 y 1. Su valor aumenta cuando hay pocos falsos positivos. Mide que las instancias clasificadas como clase C sean realmente de la clase C. Se calcula de la siguiente forma:

$$precisión(C) = \frac{\#\,de\,verdaderos positivos}{\#\,de\,verdaderos positivos + \#\,de\ \ falsos\ positivos}$$

- *F-measure*: mide la performance de un clasificador. Se calcula de la siguiente forma:

$$fmeasure = 2 * \frac{precision * recall}{precision + recall}$$

- *Matthews Correlation Coefficient (MCC)*: mide la calidad de clasificaciones binarias, es decir sobre dos clases. Devuelve un valor entre -1 y 1. Un coeficiente de 1 representa una predicción perfecta, 0 una no mejor que una predicción aleatoria y -1 indica una discrepancia total entre la predicción y la realidad. Se calcula de la siguiente forma:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- *Training time (TT)*: mide el tiempo requerido para realizar el entrenamiento del clasificador.
- *Evaluation time (ET)*: mide el tiempo requerido para realizar la evaluación del clasificador, utilizando la técnica *10-fold cross-validation*.

El objetivo del trabajo es encontrar el clasificador que tenga mayor *Exactitud*, es decir aquel que clasifique correctamente la mayor cantidad de instancias. Sin embargo, también es crucial para este trabajo que el número de *Falsos Positivos* sea el menor posible. Esto se debe a que en el filtrado de Spam es más perjudicial tener *Falsos Positivos* (mensajes legítimos marcados como Spam) que tener *Falsos Negativos*, ya que el usuario corre el riesgo de perder un mensaje importante.

Con respecto a las métricas más generales de evaluación de un clasificador, *F-measure* no tiene en cuenta los *Verdaderos Negativos* por lo que *MCC* es preferible para medir la performance de los clasificadores binarios. *MCC* es una de las mejores métricas que describen la matriz de confusión en un sólo valor. También, *MCC* es mejor que la métrica de *Exactitud* cuando las clases son de diferentes tamaños. Por ejemplo, asignar todas las instancias a la clase de mayor tamaño nos dará un número mayor de predicciones correctas pero generalmente no es una clasificación útil.

En conclusión, en este informe sólo se utilizarán las métricas *MCC*, *Exactitud*, *Falsos Positivos*, *Falsos Negativos*, *Training time* y *Evaluation time* para comparar los distintos clasificadores. Los valores de todas las métricas presentadas, menos las de tiempo, fueron obtenidos utilizando las herramientas brindadas por Weka. Sólo se utilizó un subconjunto de estos valores para poder realizar un análisis más entendible.

4.1. Prueba inicial

Se propuso como prueba inicial una configuración básica de los filtros y los clasificadores con el fin de tener una base o punto de partida sobre el cual analizar posibles mejoras.

Una vez ya transformado el dataset a un archivo ARFF, se continuó con su carga para utilizar todos estos datos para la etapa de entrenamiento y evaluación.

Como se explicó anteriormente, a los datos se les aplicó el filtro `StringToWordVector` con las opciones descriptas. El tokenizer utilizado en esta prueba fue *TK-Default* y `wordsToKeep` se probó también con el valor por defecto (1.000).

Para el entrenamiento y la evaluación se creó un `FilteredClassifier` al cual se le seteó el filtro mencionado y se fue cambiando el clasificador con el fin de probar los algoritmos seleccionados. Los resultados son los siguientes:

Clasificador	*MCC*	*ACCY %*	*FP*	*FN*	*TT*	*ET*
SMO_W1000_TK-Default	0,923	98,2418	32	66	12s	1m31s
NBayes_W1000_TK-Default	0,891	97,5063	57	82	5s	59s
PART_W1000_TK-Default	0,870	97,0219	69	97	3m39s	28m1s
IB1_W1000_TK-Default	0,792	95,4431	3	251	1s	44s
IB3_W1000_TK-Default	0,677	93,2185	0	378	1s	50s
IB5_W1000_TK-Default	0,605	91,9627	0	448	1s	51s

Figura 4: Resultados de la prueba inicial

Como puede observarse, el mejor clasificador en esta prueba es SMO. Este algoritmo es el que clasificó la menor cantidad de instancias incorrectas, 98 de 5574. En cuanto a los *Falsos Positivos*, IBk fue el que mostró los valores más bajos, incluso cero para k = 3 y 5, pero esto se debe a que el clasificador envió la mayoría de las instancias a la clase *ham*, lo cual derivó en un número alto de *Falsos Negativos*. Por este tipo de casos es que se decidió ordenar los algoritmos en base al valor de *MCC*, el cual denota la performance general de cada algoritmo.

En cuanto a la eficiencia, llama la atención el tiempo requerido por el clasificador PART en relación al resto, ya que necesitó de 31m40s para finalizar ambas etapas. De los otros clasificadores, SMO fue el más lento pero en menor medida, lo cual es justificable dada su exactitud.

4.2. Prueba ampliando el conjunto de atributos

En la prueba inicial el filtro `StringToWordVector` tenía el valor por defecto para `wordsToKeep` (1.000). En esta prueba se configuró `wordsToKeep` con un valor muy alto: 1.000.000, con el objetivo de incluir todas las palabras del dataset como atributos.

En la prueba anterior la cantidad de atributos era de 1.077, en esta prueba el conjunto de atributos es de 9.166. Seguramente los tiempos de procesamiento sean mayores pero se espera obtener una mejor precisión en los algoritmos.

Los resultados de las evaluaciones son los siguientes:

Clasificador	***MCC***	***ACCY %***	***FP***	***FN***	***TT***	***ET***
SMO_W1000000_TK-Default	0,945	98,7442	3	67	27s	2m38s
NBayes_W1000000_TK-Default	0,893	97,5422	56	81	42s	7m31s
PART_W1000000_TK-Default	0,871	97,0398	72	93	31m42s	3h52m48s
IB1_W1000000_TK-Default	0,778	95,1561	0	270	4s	1m36s
IB3_W1000000_TK-Default	0,624	92,2856	0	430	5s	1m46s
IB5_W1000000_TK-Default	0,559	91,2092	0	490	5s	1m55s

Figura 5: Resultados de la prueba ampliando el conjunto de atributos

Con los datos recabados se puede afirmar que se logró una mejora importante sobre el algoritmo SMO, el cual hasta aquí ha tenido la mayor performance y calidad. Se aumentó el *MCC* y la *Exactitud*, y se redujo la cantidad de *Falsos Positivos* de 32 a 3, lo cual es el objetivo de este trabajo.

Con respecto a NaiveBayes y PART la mejora fue mínima. Sin embargo, en el algoritmo PART el tiempo de procesamiento se amplió 8 veces, tardando casi 4 horas en completar ambas etapas, y aumentaron los *Falsos Positivos* por lo que no se considerará el cambio planteado como una mejora en este algoritmo. Así mismo, los algoritmos IBk tampoco mostraron buenos resultados ya que todas sus métricas decrecieron.

4.3. Prueba con tokenizer *TK-Complete*

En esta prueba se modificó el delimitador del tokenizer con el objetivo de optimizar la separación de tokens. La cantidad de atributos se redujo a 1.006 y 8.817 para `wordsToKeep` 1.000 y 1.000.000, respectivamente. Como ya se explicó, este tokenizer remueve símbolos innecesarios que están presentes en el dataset. En consecuencia, los tokens que contenían estos símbolos serán divididos permitiendo mejorar el peso de aquellos tokens que ya existían y coinciden con los tokens resultantes de la división. Esta es la principal causa de la reducción de atributos.

Se realizaron pruebas para ambos valores de `wordsToKeep` con el fin de observar el comportamiento de los distintos algoritmos. Los resultados de las evaluaciones son los siguientes:

Clasificador	*MCC*	*ACCY %*	*FP*	*FN*	*TT*	*ET*
SMO_W1000_TK-Complete	0,924	98,2598	36	61	11s	1m34s
NBayes_W1000_TK-Complete	0,894	97,5601	59	77	5s	58s
PART_W1000_TK-Complete	0,862	96,8066	86	92	3m47s	33m22s
IB1_W1000_TK-Complete	0,795	95,5149	11	239	1s	55s
IB3_W1000_TK-Complete	0,687	93,3979	0	368	1s	57s
IB5_W1000_TK-Complete	0,616	92,1421	0	438	2s	59s

Figura 6: Resultados de la prueba con tokenizer TK-Complete y wordsToKeep 1.000

Clasificador	*MCC*	*ACCY %*	*FP*	*FN*	*TT*	*ET*
SMO_W1000000_TK-Complete	0,946	98,7621	3	66	33s	2m27s
NBayes_W1000000_TK-Complete	0,896	97,6139	57	76	41s	7m8s
PART_W1000000_TK-Complete	0,858	96,7169	90	93	26m36s	3h35m51s
IB1_W1000000_TK-Complete	0,786	95,3175	1	260	4s	1m38s
IB3_W1000000_TK-Complete	0,634	92,4471	0	421	4s	1m44s
IB5_W1000000_TK-Complete	0,560	91,2271	0	489	4s	1m46s

Figura 7: Resultados de la prueba con tokenizer TK-Complete y wordsToKeep 1.000.000

Para `wordsToKeep` 1.000, SMO mejoró en base a la prueba anterior de W1000, NaiveBayes superó las pruebas anteriores y los IBk tuvieron una leve mejora. Sin embargo, en todos los casos cayeron los *Falsos Positivos*. Esto puede deberse a que el conjunto de atributos se redujo, lo que generó que el clasificador tenga mayores problemas para predecir la clase de un mensaje.

Para `wordsToKeep` 1.000.000, SMO tuvo una mínima mejora, logrando así el mejor clasificador hasta aquí. Esto se debe a que disminuyó en un valor el número de *Falsos Negativos*. El algoritmo NaiveBayes y los IBk también mostraron avances, pero estos no han sido muy significativos aún.

Con el algoritmo PART no se ha podido lograr una mejora desde la prueba inicial y los tiempos de entrenamiento y evaluación aumentaron excesivamente. Por ello, de aquí en adelante, las pruebas no se ejecutarán más sobre este algoritmo.

Por otra parte, de los algoritmos IBk, el clasificador con `k` = 1 tuvo en todos los casos mejor performance que los clasificadores con `k` = 3 y 5. Por lo tanto, sólo se continuarán las pruebas sobre IBk con `k` = 1.

También, las pruebas siguientes se realizarán sólo para `wordsToKeep` 1.000.000 ya que ha dado mejores resultados sobre los algoritmos con mayor precisión.

4.4. Prueba con tokenizer *TK-Numbers*

En esta prueba se modificó nuevamente el delimitador del tokenizer con la idea de optimizar aún más la separación de tokens. La cantidad de atributos se redujo a 7.828. Como se mencionó anteriormente, al delimitador *TK-Complete* se agregan los números con el fin de eliminar la gran cantidad de tokens que son sólo números y separar aquellos que contienen números. En consecuencia, se aumentará el peso de aquellos tokens que ya existían y coinciden con los tokens resultantes de la división.

Los resultados de las evaluaciones son los siguientes:

Clasificador	*MCC*	*ACCY %*	*FP*	*FN*	*TT*	*ET*
SMO_W1000000_TK-Numbers	0,951	98,8698	5	58	23s	2m16s
NBayes_W1000000_TK-Numbers	0,921	98,188	39	62	37s	6m22s
IB1_W1000000_TK-Numbers	0,810	95,8199	5	228	4s	1m38s

Figura 8: Resultados de la prueba con tokenizer TK-Numbers

Como puede observarse, los algoritmos tuvieron la mayor exactitud, performance y eficiencia de todas las pruebas realizadas hasta aquí. Los algoritmos NaiveBayes e IB1 mostraron un aumento en la *Exactitud* del 0,5%, dejando a NaiveBayes por arriba del 98% de *Exactitud* y reduciendo sus *Falsos Positivos* de 57 a 39. Por lo tanto, en esta prueba NaiveBayes fue el más beneficiado teniendo en cuenta los objetivos de este trabajo.

Por otro lado, el algoritmo SMO presenta un problema de tradeoff. Todas sus métricas aumentaron excepto por los *Falsos Positivos*. Se redujo en 8 la cantidad de *Falsos Negativos* pero aumentó en 2 los *Falsos Positivos*. Si tenemos en cuenta el total de mensajes, el porcentaje de cambio de los *FP* es muy bajo. Sin embargo, el objetivo es conseguir el menor número de *FP*. Llevado esto al uso del clasificador, es preferible que entren un par de mensajes Spam a la bandeja de entrada, a que un mensaje importante o de emergencia sea clasificado como Spam. En conclusión, no se tomará en cuenta esta prueba como mejora al algoritmo SMO.

4.5. Prueba con *Boosting*

En esta prueba se aplicará la técnica de *Boosting* sobre los algoritmos SMO, NaiveBayes e IB1. Como se explicó, el método *Boosting* está basado en la construcción de sucesivos clasificadores sobre el conjunto de entrenamiento que se va modificando en función de los errores cometidos por el clasificador anterior. Se espera obtener mayor exactitud en los algoritmos más débiles con el costo de un aumento en los tiempos de procesamiento.

Los resultados de las evaluaciones son los siguientes:

Clasificador	***MCC***	***ACCY %***	***FP***	***FN***	***TT***	***ET***
SMO_W1000000_TK-Complete_Boosting	0,947	98,7801	3	65	29s	5m55s
NBayes_W1000000_TK-Complete_Boosting	0,926	98,2957	38	57	44m42s	8h1m
IB1_W1000000_TK-Complete_Boosting	0,786	95,3175	1	260	47s	6m52s

Figura 9: Resultados de la prueba con Boosting y TK-Complete

Clasificador	***MCC***	***ACC %***	***FP***	***FN***	***TT***	***ET***
SMO_W1000000_TK-Numbers_Boosting	0,949	98,8339	6	59	3m25s	18m24s
NBayes_W1000000_TK-Numbers_Boosting	0,933	98,4571	38	48	39m47s	5h42m6s
IB1_W1000000_TK-Numbers_Boosting	0,810	95,8199	5	228	45s	6m29s

Figura 10: Resultados de la prueba con Boosting y TK-Numbers

Los resultados de aplicar *Boosting* muestran que el clasificador SMO tuvo una leve mejora sobre aquel en el que no se usó la técnica. Se logró disminuir en uno la cantidad de *Falsos Negativos* para *TK-Complete*. Si bien los tiempos requeridos en las etapas se duplicaron, esta prueba se considera la mejor para SMO.

NaiveBayes obtuvo una mayor mejora para ambos tokenizers, más marcada para *TK-Complete* y en menor medida para *TK-Numbers*. De su mejor versión previa, se llegó a reducir en uno los *Falsos Positivos* y en 14 los *Falsos Negativos*. Además, su *Exactitud* aumentó cerca del 0,3%. Cabe destacar que la eficiencia decayó ampliamente, necesitando alrededor de 9 horas y más de 6 horas para realizar las pruebas mencionadas.

Por último, el clasificador IB1 no mostró ningún cambio en relación a los resultados previos. Es decir, el *Boosting* no afectó el desempeño del algoritmo IB1.

4.6. Pruebas insatisfactorias

Como ya se ha expresado, debido a que la longitud de los mensajes SMS es corta, la utilización de herramientas de reducción de dimensionalidad tiende a afectar la precisión en el filtrado de Spam. Sólo con el fin de mostrar que este es el caso se realizó una breve evaluación de los

resultados de utilizar *stop word removal*, *word stemming* y *attribute selection*.

Para *stop word removal* se buscó una lista de stop words en Internet y se cargó al filtro `StringToWordVector`.

Para *word stemming* se utilizó el stemmer `LovinsStemmer`, también dentro del filtro `StringToWordVector`.

Para *attribute selection* se definió un `MultiFilter` para poder agregar el filtro `AttributeSelection`, cuyo evaluador seleccionado fue `InfoGainAttributeEval` y el buscador fue `Ranker` con el `threshold` en cero.

Otra prueba que se realizó fue la utilización de las transformaciones *TF-IDF* en el filtro `StringToWordVector`.

Los resultados de las evaluaciones son los siguientes:

Clasificador	***MCC***	***ACC %***	***FP***	***FN***	***TT***	***ET***
SMO_W1000000_TK-Complete_TF-IDF	0,945	98,7442	3	67	-	-
SMO_W1000000_TK-Complete_Stemming	0,945	98,7442	6	64	-	-
SMO_W1000000_TK-Complete_AttSelection	0,937	98,5648	5	75	1m9s	8m53s
SMO_W1000000_TK-Complete_StopWords	0,927	98,3315	7	86	-	-
NBayes_W1000000_TK-Complete_AttSelection	0,896	97,596	59	75	1m18s	10m24s
IB1_W1000000_TK-Complete_AttSelection	0,780	95,2099	2	265	1m10s	9m8s

Figura 11: Resultados de las pruebas insatisfactorias

Como puede observarse, ninguna de las pruebas fue capaz de mejorar los resultados obtenidos previamente para *TK-Complete* y `wordsToKeep` 1.000.000 (sección 4.3). De hecho, la utilización de estas herramientas hizo decaer los valores de las métricas en todos los casos. Por lo tanto, se confirma que la reducción de dimensionalidad en este caso de estudio es contraproducente.

4.7. Resultados finales

En esta sección se agruparán los mejores resultados obtenidos para cada uno de los algoritmos de clasificación previamente evaluados, teniendo en cuenta las decisiones de tradeoff realizadas. A continuación se detalla la lista de clasificadores con sus métricas:

Clasificador	***MCC***	***ACC %***	***FP***	***FN***	***TT***	***ET***
SMO_W1000000_TK-Complete_Boosting	0,947	98,7801	3	65	29s	5m55s
NBayes_W1000000_TK-Numbers_Boosting	0,933	98,4571	38	48	39m47s	5h42m6s
PART_W1000_TK-Default	0,870	97,0219	69	97	3m39s	28m1s
IB1_W1000000_TK-Numbers	0,810	95,8199	5	228	4s	1m38s

Figura 12: Resultados finales

De los resultados finales puede concluirse que las distintas características y opciones seleccionadas para las pruebas contribuyeron a mejorar los diferentes algoritmos de clasificación. Excepto PART, el resto de los algoritmos fueron perfeccionados gracias a la ampliación del conjunto de atributos con `wordsToKeep` 1.000.000 y a la optimización en la separación de tokens con los delimitadores *TK-Complete* y *TK-Numbers*. A su vez, los mejores algoritmos, SMO y NaiveBayes, se vieron beneficiados por la técnica de *Boosting*. Por otro lado, PART no mostró mejoras significativas con los cambios efectuados.

5. Aplicación

Como complemento del análisis de los algoritmos, se desarrolló una aplicación en Java la cual permite probar los distintos clasificadores a la hora de filtrar mensajes de texto SMS, así como entrenar y evaluar cada uno de los clasificadores presentados en este trabajo.

Esta aplicación fue realizada con el objetivo de poder evaluar los clasificadores de una manera amigable y sencilla, trasladando así todo el conocimiento adquirido a una aplicación real y muy fácil de usar.

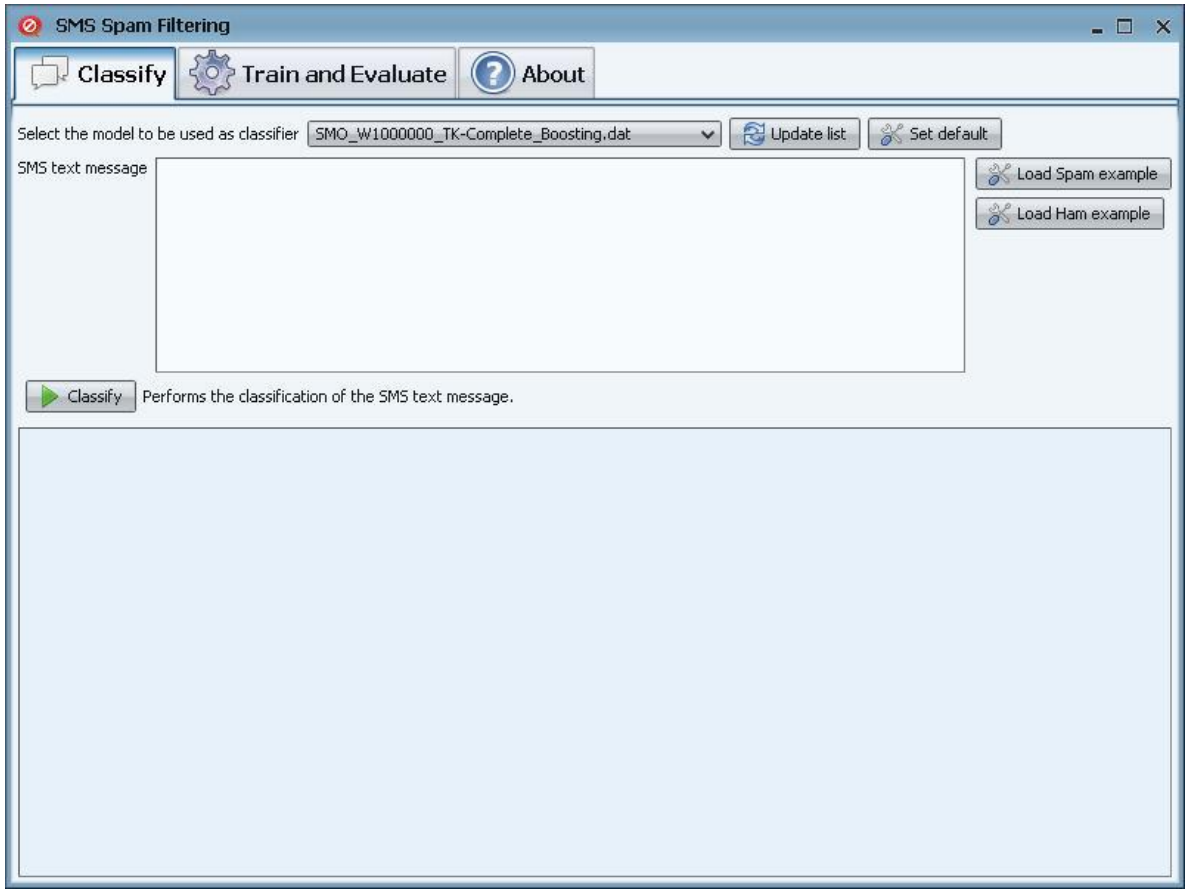

Figura 13: Interfaz de la aplicación SMS Spam Filtering

Como se puede observar en la figura 13, la estructura general de la aplicación está compuesta por tres solapas: Classify, Train and Evaluate y About.

La primera solapa, Classify, provee las funcionalidades necesarias para decidir si un mensaje de texto SMS es Spam o no. Para ello se debe seleccionar el modelo del clasificador que se quiere evaluar, luego se introduce el texto del mensaje en el campo "SMS text message" y a continuación se presiona el botón "Classify". El resultado de la clasificación se presenta de la siguiente forma:

Figura 14: Respuesta de la clasificación de un SMS

Cabe aclarar que debido a que el dataset está conformado por mensajes en inglés, el texto para las pruebas también debe estar en ese idioma. Para facilitar los pruebas, se agregaron dos botones: "Load Spam example" y "Load Ham example", los cuales completan el campo del mensaje con un SMS en inglés. Con el fin de que estas pruebas sean más prácticas, se buscaron en Internet mensajes de texto auténticos. Además, se corroboró que estos mensajes no estuviesen ya en el dataset para que la evaluación sea más realista y no se vea beneficiada por los datos utilizados para el entrenamiento de los clasificadores.

Con respecto a la selección del modelo de prueba, se provee una lista formada por todos los archivos .dat que se encuentran en la misma carpeta donde se ejecuta el programa. Cada uno de estos archivos representa un `FilteredClassifier`, los cuales fueron creados con anterioridad desde la solapa "Train and Evaluate". El botón "Update list" permite actualizar esta lista en caso de que un nuevo modelo se haya generado y el mismo quiera probarse. El botón "Set default" selecciona el algoritmo de clasificación que mejores resultados dio en las evaluaciones realizadas en la sección 4: SMO_W1000000_TK-Complete_Boosting.

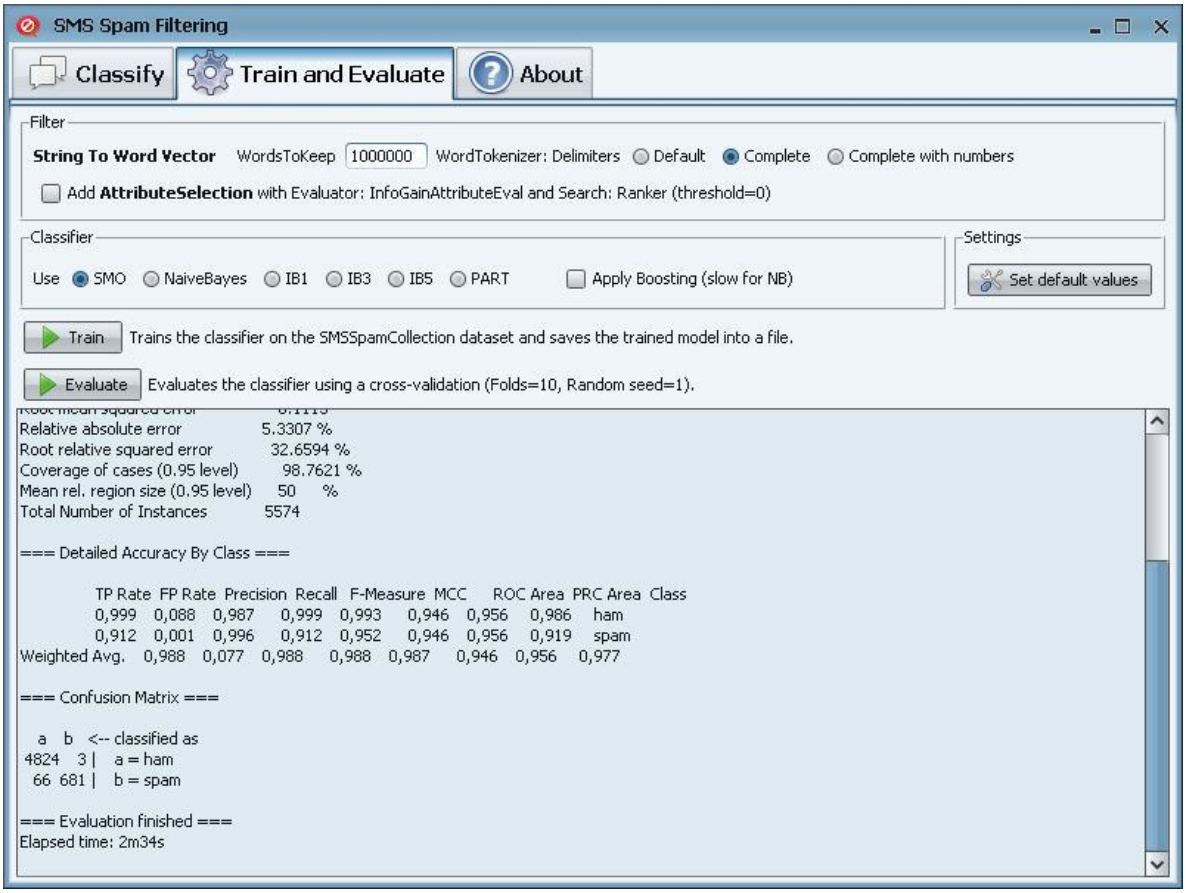

Figura 15: Solapa Train and Evaluate

Como muestra la figura 15, la solapa Train and Evaluate permite configurar las características del filtrado de los datos, seleccionar el algoritmo de clasificación y ejecutar el entrenamiento o la evaluación de un clasificador.

Las opciones del filtrado son aquellas que se presentaron en la sección 3. Es decir, se puede configurar el valor de `wordsToKeep` para el filtro `StringToWordVector`, se puede seleccionar entre los tokenizers *TK-Default*, *TK-Complete* y *TK-Numbers*, y se puede añadir el filtro `AttributeSelection`.

En tanto, para el clasificador se puede seleccionar entre SMO, NaiveBayes, IB1, IB3, IB5 y

PART. Además, se puede optar por utilizar la técnica de *Boosting*.

Si se quiere entrenar un clasificador, se debe presionar el botón "Train". Aquí, el programa lo que hace es cargar el dataset que se encuentra en el archivo SMSSpamCollection.arff. Luego se crea el clasificador `FilteredClassifier` en base a las opciones seleccionadas en la interfaz. A continuación, se entrena este clasificador con los datos cargados del dataset. Una vez que termina el entrenamiento, se guarda este `FilteredClassifier` ya entrenado en un archivo .dat. El nombre de este archivo se forma en base a las opciones seleccionadas para poder diferenciar los distintos modelos fácilmente. Este modelo se guarda para poder reutilizarlo cuando se quiere clasificar un nuevo SMS, desde la solapa Classify. Así, se ahorra el tiempo de entrenamiento cada vez que se quiere realizar una prueba de un clasificador.

Si se quiere evaluar un clasificador, se debe presionar el botón "Evaluate". En este caso el programa también debe cargar el dataset desde SMSSpamCollection.arff y crear el clasificador `FilteredClassifier` de la misma forma que para el entrenamiento. Finalmente, se ejecuta la evaluación *10-fold cross-validation* con los datos cargados del dataset.

Los resultados del entrenamiento y la evaluación de un clasificador se muestran en el log, ubicado en la parte inferior de la interfaz. En este log se indica en todo momento el estado de la tarea en ejecución. Así mismo, ni bien comienza una tarea se muestra en el log el tiempo estimado para realizarla. Este tiempo es aproximado y se corresponde con los tiempos de entrenamiento y evaluación presentados en las pruebas de la sección 4. Cuando una tarea finaliza, también se muestra el tiempo transcurrido, por si se quieren efectuar análisis de eficiencia de los algoritmos.

Las tareas de Clasificar, Entrenar y Evaluar se pueden cancelar en cualquier momento. Esta funcionalidad se implementó más que nada para poder terminar aquellos procesos que puedan requerir de mucho tiempo de procesamiento. Como complemento, al mostrar el tiempo estimado para una tarea, si este es muy alto se sugiere al usuario que el mismo puede cancelar la tarea.

Por último, con la aplicación se realizaron pruebas de clasificación sobre los mensajes de texto SMS seleccionados como ejemplos para el programa. Estos mensajes son:

Spam: *U requested daily inspirational texts. Sub service is $9.99/mo + Msg&Data rates may apply. Reply STOP to cancel. Reply HELP for HELP. Enter on web: 6082*

Ham: *When you give importance to people they think that you are always free but they don't understand that you make yourself available for them every time*

Se probó cada uno de los modelos disponibles en los archivos .dat y se pudo corroborar parte de los resultados encontrados en la sección 4. En resumen, todos los modelos clasificaron el mensaje *ham* correctamente, es decir como *ham*. Esto es importante, ya que en otras palabras ninguno generó *Falsos Positivos*, lo cual es uno de los propósitos de este trabajo. En cuanto al mensaje *spam*, los algoritmos de clasificación IBk lo clasificaron incorrectamente como *ham*. Como se vio en las evaluaciones, este tipo de clasificadores tuvo un número alto de *Falsos Negativos* y esta prueba también lo demostró. Por otra parte, el resto de los algoritmos: SMO, NaiveBayes y PART, clasificaron el mensaje *spam* correctamente, es decir como *spam*.

6. Conclusiones

En este trabajo se han podido aplicar satisfactoriamente las técnicas aprendidas en el curso de Minería de Datos Web. Se logró adaptar el conjunto de datos *SMS Spam Collection Data Set* para su posterior utilización con las herramientas provistas por Weka. Se definieron luego los puntos a analizar en las etapas de pre-procesamiento, clasificación y pos-procesamiento de los datos.

En una primera instancia, se detallaron las características a examinar sobre el filtrado de los datos. Además, se especificaron cuatro algoritmos de clasificación a estudiar, con diferentes técnicas y configuraciones.

Seguidamente, se realizó una evaluación minuciosa y detallada de los clasificadores en base a las distintas opciones utilizadas para las pruebas. Para ello, se tuvieron en cuenta diferentes métricas de efectividad y eficiencia, las cuales fueron formalmente definidas. De las 39 pruebas realizadas, se presentaron las mejores combinaciones para los cuatro algoritmos de clasificación. De estos resultados finales se concluyó que las distintas características y opciones seleccionadas para las sucesivas pruebas contribuyeron a mejorar los diferentes algoritmos de clasificación.

Finalmente, se desarrolló y se presentó una aplicación capaz de predecir si un cierto mensaje de texto SMS es Spam o no, a partir de la utilización de los algoritmos estudiados. Además, la aplicación brinda la posibilidad de entrenar y evaluar los clasificadores con las mismas configuraciones que fueron presentadas en este informe.

Apéndice

Lista completa de las 39 pruebas realizadas, ordenada por *MCC*:

Clasificador	*MCC*	*ACC %*	*FP*	*FN*	*TT*	*ET*
SMO_W1000000_TK-Numbers	0,951	98,8698	5	58	23s	2m16s
SMO_W1000000_TK-Numbers_Boosting	0,949	98,8339	6	59	3m25s	18m24s
SMO_W1000000_TK-Complete_Boosting	0,947	98,7801	3	65	29s	5m55s
SMO_W1000000_TK-Complete	0,946	98,7621	3	66	33s	2m27s
SMO_W1000000_TK-Default	0,945	98,7442	3	67	27s	2m38s
SMO_W1000000_TK-Complete_TF-IDF	0,945	98,7442	3	67	-	-
SMO_W1000000_TK-Complete_Stemming	0,945	98,7442	6	64	-	-
SMO_W1000000_TK-Complete_AttSelection	0,937	98,5648	5	75	1m9s	8m53s
NBayes_W1000000_TK-Numbers_Boosting	0,933	98,4571	38	48	39m47s	5h42m6s
SMO_W1000000_TK-Complete_StopWords	0,927	98,3315	7	86	-	-
NBayes_W1000000_TK-Complete_Boosting	0,926	98,2957	38	57	44m42s	8h1m
SMO_W1000_TK-Complete	0,924	98,2598	36	61	11s	1m34s
SMO_W1000_TK-Default	0,923	98,2418	32	66	12s	1m31s
NBayes_W1000000_TK-Numbers	0,921	98,188	39	62	37s	6m22s
NBayes_W1000000_TK-Complete	0,896	97,6139	57	76	41s	7m8s
NBayes_W1000000_TK-Complete_AttSelection	0,896	97,596	59	75	1m18s	10m24s
NBayes_W1000_TK-Complete	0,894	97,5601	59	77	5s	58s
NBayes_W1000000_TK-Default	0,893	97,5422	56	81	42s	7m31s
NBayes_W1000_TK-Default	0,891	97,5063	57	82	5s	59s
PART_W1000000_TK-Default	0,871	97,0398	72	93	31m42s	3h52m48s
PART_W1000_TK-Default	0,870	97,0219	69	97	3m39s	28m1s
PART_W1000_TK-Complete	0,862	96,8066	86	92	3m47s	33m22s
PART_W1000000_TK-Complete	0,858	96,7169	90	93	26m36s	3h35m51s
IB1_W1000000_TK-Numbers	0,810	95,8199	5	228	4s	1m38s
IB1_W1000000_TK-Numbers_Boosting	0,810	95,8199	5	228	45s	6m29s
IB1_W1000_TK-Complete	0,795	95,5149	11	239	1s	55s
IB1_W1000_TK-Default	0,792	95,4431	3	251	1s	44s
IB1_W1000000_TK-Complete	0,786	95,3175	1	260	4s	1m38s
IB1_W1000000_TK-Complete_Boosting	0,786	95,3175	1	260	47s	6m52s
IB1_W1000000_TK-Complete_AttSelection	0,780	95,2099	2	265	1m10s	9m8s
IB1_W1000000_TK-Default	0,778	95,1561	0	270	4s	1m36s
IB3_W1000_TK-Complete	0,687	93,3979	0	368	1s	57s
IB3_W1000_TK-Default	0,677	93,2185	0	378	1s	50s
IB3_W1000000_TK-Complete	0,634	92,4471	0	421	4s	1m44s
IB3_W1000000_TK-Default	0,624	92,2856	0	430	5s	1m46s
IB5_W1000_TK-Complete	0,616	92,1421	0	438	2s	59s
IB5_W1000_TK-Default	0,605	91,9627	0	448	1s	51s
IB5_W1000000_TK-Complete	0,560	91,2271	0	489	4s	1m46s
IB5_W1000000_TK-Default	0,559	91,2092	0	490	5s	1m55s