UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Repositorio Digital USM **https://repositorio.usm.cl**

Tesis USM TESIS de Pregrado de acceso ABIERTO

2018-04

COMPRESIÓN DE TWEETS USANDO ALGORITMO DE HUFFMAN CON MODELOS PREDEFINIDOS

ROJAS VERA, JEREMY ISAI

http://hdl.handle.net/11673/41283

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE

“COMPRESIÓN DE TWEETS USANDO ALGORITMO DE HUFFMAN CON MODELOS PREDEFINIDOS”

JEREMY ISAI ROJAS VERA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: DIEGO ARROYUELO

ABRIL 2018

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE

“COMPRESIÓN DE TWEETS USANDO ALGORITMO DE HUFFMAN CON MODELOS PREDEFINIDOS”

JEREMY ISAI ROJAS VERA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: DIEGO ARROYUELO

PROFESOR CORREFERENTE: RICARDO ÑANCULEF

ABRIL 2018

MATERIAL DE REFERENCIA, SU USO NO INVOLUCRA RESPONSABILIDAD DEL AUTOR O DE LA INSTITUCIÓN

Agradecimientos

Primero que todo agradezco a Dios por haberme dado la oportunidad de efectuar este camino hasta la ingeniería, ya que estoy seguro de que, de él, por él y para él son Todas las cosas. Soy un convencido que si no hubiese sido por la voluntad de Dios nada de esto hubiese sido posible, ya que es a través de él que obtengo las fuerzas para superar las adversidades.

Agradezco a mis Padres por siempre estar ahí, por animarme cuando no veía una salida a un problema, por encenderme la estufa en esas frías noches de estudio, por irme a dejar a una estación de metro para llegar a la universidad o irme a buscar cuando estaba muy cansado, entre muchas otras cosas, por tenerme paciencia, comprenderme y apoyarme cada día de este viaje por la USM.

Agradezco también a mi familia por nunca dejar de confiar en mí, aunque muchas veces la meta se veía lejos, siempre confiaron en que podría superar todo lo que viniera por delante.

También agradezco a mis compañeros de generación, ya que creo que esto no se hace solo, se hace con compañerismo y eso vi durante todos estos años, vi ingenieros dispuestos ayudar cuando era difícil resolver problemas, vi los mejores profesores cuando la materia era difícil de comprender y encontré amigos con los cuales compartir.

Por último, quiero agradecer a mi iglesia, porque en cada paso que daba, sé que ellos oraban por mí y pedían que Dios me ayudara en este camino. Iglesia, sentí sus oraciones.

Resumen

El aumento en el uso de redes sociales ha provocado que nueva información se genere de forma masiva día a día en Internet. Este hecho causa que los textos cortos como post en redes sociales, comentarios, tweets, discuciones, etc. sean cada vez mas recurrentes y por consecuencia requieren gran capacidad para su almacenamiento.

Este trabajo de memoria tiene como finalidad el estudio del comportamiento de un compresor clásico aplicado en textos pequeños, proponiendo un método para que la tasa de compresión en estos textos disminuya.

Huffman es un algoritmo de compresión clásico que posee una tabla de frecuencias de símbolos que codifica y decodifica textos. Esta tabla al anexarla a textos cortos provoca que la compresión no sea eficiente.

La generalización de estas tablas de frecuencias se denominarán modelos predefinidos de compresión, que tienen como objetivo la extracción de dichas tablas de los archivos codificados y con ello reducir las tasas de compresión en textos diminutos.

Keywords: Compresión, Huffman, Algoritmo, Tweet, Modelos predefinidos, “Cluster”.

Abstract

The increase in the usage of social media networks has produced that new information starts to generate massively everyday on Internet. This fact causes that small texts like posts in social media networks, posts, tweets, discussions, among others, are every time more recurrent and consequently, they need a great capacity for their storage.

This work has as target the study of the behavior of a classic compressor applied in small texts, proposing one method in which the compression rate in this kind of texts decreases.

Huffman is a classic compression algorithm that has a frequency table of symbols that encodes and decodes texts. This table inside of the small text produces inefficient compression.

The generalization of this frequency tables will be denominated predefined models of compression, that have as objective the extraction of those tables from the encoded archives and, in that way, the decrease of the compression rate in small texts.

Keywords: Compression, Huffman, Algorithm, Tweet, Predefined Models, Cluster.

Índice de Contenidos

Agradecimientos III

Resumen IV

Abstract V

Índice de Contenidos VI

Lista de Tablas VIII

Lista de Figuras IX

Glosario XI

Introducción 1

1. Definición del Problema 3

2. Estado del Arte y Conceptos Fundamentales 5

2.1. Definición de Tasa de Compresión 5

2.2. Compresores de Texto Corto 6

2.3. Algoritmo de Huffman 13

2.4. Otros Compresores Tradicionales 17

3. Compresión de tweets usando algoritmo de Huffman con modelos predefinidos de compresión **18**

3.1. Propuesta 18

3.2. Implementación 22

3.2.1. Modelos Estocásticos 23

3.2.2. Modelos mediante Clustering 25

3.2.3. Cotejo entre modelo predefinidos de compresión y tweet 26

4. Resultados **28**

4.1. Descarga de conjunto de datos para pruebas 28

4.2. Creación de modelos predefinidos de compresión 30

4.3. Compresión de Tweets 32

4.4. Huffman con modelos predefinidos de compresión v/s SMAZ 39

5. Conclusiones y Trabajo A Futuro **43**

Bibliografía **46**

Índice de cuadros

2.1. Tablas comparativas de algoritmos BWT con Huffman y LZW, con diferentes textos de tamaño corto. 8

2.2. Tabla comparativa de rendimiento entre TS03.42 y SMAZ. 10

2.3. Tabla de rendimiento de *SMAZ*, en donde figura la compresión de tweets . . 10

2.4. Tabla de rendimiento de *stats3*, contrarrestado con *SMAZ* y *TS03.42*. 12

2.5. Codificación binaria de Huffman para símbolos de texto *"ata la jaca a la estaca"*. 16

2.6. Codificación binaria de Huffman para texto *"ata la jaca a la estaca"* 16

4.1. Comparación de los procesos de creación de los modelos predefinidos de compresión 32

4.2. Tabla comparativa de rendimiento de los dos métodos que empíricamente mostraron resultados positivos. 39

Índice de figuras

2.1. Tasa de compresión de Algoritmo LCC, para textos cortos pre definidos. . . 7

2.2. Ejemplo de transcodificación usada en algoritmo *b64pack*. 8

2.3. Promedio de velocidad de compresión en función de los algoritmos *b64pack*, compress, gzip y bzip2. 9

3.1. Gráfico comparativo de rendimiento en la compresión de textos cortos de compresores clásicos. 19

3.2. Gráfico comparativo de rendimiento en la compresión de tweets en grupos por diferentes compresores clásicos. 20

3.3. Diagrama explicativo de elección de modelos estocásticos para la compresión de tweets. 24

3.4. Diagrama explicativo de elección de modelos mediante “clustering” para la compresión de tweets. 26

4.1. Distribución de tweets en función de su tamaño en disco. 29

4.2. Gráfico comparativo de las tasas de compresión media. 34

4.3. Gráfico comparativo entre tamaños promedios de compresiones de tweets al usar diferentes métodos de compresión. 35

4.4. Gráfico comparativo entre tasas promedios de compresión de tweets al usar diferentes métodos de compresión. 36

4.5. Gráficos comparativos de desempeño de compresiones con resultados positivos. 38

4.6. Gráfico comparativo de tamaño de compresión media en función del tamaño del tweet sin comprimir, incluyendo el algoritmo de compresión Smaz. . . . 41

4.7. Gráfico comparativo de tamaño de compresión media en función del tamaño del tweet sin comprimir, incluyendo el algoritmo de compresión Smaz, para textos menores que 150 bytes. 42

Glosario

- Algoritmo: Conjunto de instrucciones que permiten llegar a la solución de algún tipo de problema. Un algoritmo, habitualmente se traduce en un programa de computador.
- API: Interfaz de programación de aplicaciones que provee un servicio de consulta de datos.
- Estocástico: Proceso que se realiza de manera aleatoria.
- Heurística: Método diferente que hace que un procedimiento se ajuste a un objetivo particular.
- Iteratición: Reiteración de un procedimiento.
- Ratio de Compresión: Tasa que relaciona el tamaño anterior y posterior al aplicar un algortimo de compresión.

Introducción

La compresión de texto es un área que está abordada de manera amplia en la actualidad. Sin embargo, ésta es aplicada en su mayoría a textos de tamaño grande, ya que dichos textos son los que ocupan mayor espacio en disco por sí solos.

Sin embargo, hoy en día los textos pequeños son muy comunes, como los SMS en teléfonos celulares, comentarios y estados en facebook o los populares tweets en Twitter. Si bien estos textos son de tamaño pequeño y no ocupan gran espacio de memoria, al dimensionar los millones de tweets que se almacenan día a día, hacen necesario recurrir a algún método de compresión para el gran número de estos textos, de modo que el almacenamiento y la transmisión de ellos sea eficiente.

Por lo anterior, se quiere estudiar el comportamiento de compresores normalmente usados en textos grandes aplicados a textos pequeños, usando alguna heurística en particular que haga que el rendimiento al comprimir mejore en comparación al compresor en forma pura.

Se pretende estudiar dichos comportamientos usando un método empírico, mediante la experimentación en la compresión de tweets en una localidad específica del mundo.

El método de compresión sometido a estudio es el algoritmo de Huffman, que para su funcionamiento ocupa un modelo del texto creado a partir de su tabla de frecuencia de símbolos, anexada al archivo reducido, que se usa para la compresión y descompresión del mismo. La

heurística a utilizar es la selección de los modelos más representativos de un conjunto de estudio, con el objetivo de no integrar el modelo al archivo comprimido.

Cabe señalar que existen algoritmos de carácter privado y público que comprimen de manera efectiva textos cortos como tweets, pero esta memoria no pretende de ninguna manera superar el rendimiento de ellos, sino que se pretende estudiar el uso de compresores clásicos aplicados a textos para los cuales no están hechos, usando alguna modificación como modelos predefinidos de compresión.

Por lo tanto, esta memoria posee los siguientes objetivos principales:

- Probar el comportamiento del algoritmo de Huffman aplicado a textos pequeños, en particular, tweets.
- Buscar árboles de compresión de Huffman generales para un conjunto de tweets particular, con el propósito de obtener modelos de las distribuciones de frecuencia para un conjunto de textos dado.
- Usar el algoritmo de compresión de Huffman con los modelos generalizados encontrados con anterioridad, de modo de no incluir el árbol de compresión en el archivo comprimido.
- Analizar y comparar resultados de la aplicación del algoritmo de Huffman con modelos predefinidos comparándolo con algoritmo de Huffman sin modificación y con compresores de texto creados para textos cortos.

Capítulo 1

Definición del Problema

El avance de la tecnología en la vida cotidiana hizo posible el surgimiento de una era digital, en donde millones de personas están conectadas mediante dispositivos móviles e Internet.

Esto ha producido que el tránsito de información a través de Internet vaya en constante aumento, lo que es impulsado por el surgimiento de los llamados *Medios Sociales*, que abarcan un grupo de usuario considerable en la Internet.

Tomando en consideración que mucha de la información compartida en la actualidad está vinculada con los medios sociales, se observa como resultado que tanto los mensajes enviados, como los recibidos y guardados son en su mayoría textos cortos, incluso la red social *Twitter* acepta en cada publicación un máximo de 140 caracteres (280 desde noviembre de 2017).

Los mensajes y/o discusiones que se dan en la red social *facebook* también poseen esta característica: son cortos. De igual manera los mensajes que se envían y/o publican en otras redes sociales como *whatsapp*, *instagram*, *telegram*, *snapchat*, *youtube* y otras tienen la misma propiedad: son publicaciones acotadas y pequeñas, cuando se habla de información de tipo texto.

Señalado lo anterior, todas estas aplicaciones necesitan, de alguna manera, almacenar sus datos, y por el gran volumen de ellos requieren servidores de gran envergadura e incluso, las bases de datos de algunas de las compañías son consideradas de tipo big data.

Si analizamos la red social Twitter, con cada mensaje (tweet) de a lo más 140 caracteres (antes de noviembre de 2017), en noviembre de 2016, por cada segundo la red social recibió en promedio 6.000 tweets[12]. Esta cifra indica la gran cantidad de mensajes de 140 caracteres que se almacenan segundo tras segundo en las bases de datos Twitter.

Si se piensa en reducir el espacio ocupado por estos billones de textos pequeños almacenados en las bases de datos de las compañías descritas, se hace necesario pensar en algún algoritmo que sea capaz de comprimir, de modo que la transmisión y guardado de todos los datos sea de manera eficaz y eficiente.

Los algoritmos de compresión de texto más populares están construidos para comprimir texto de gran tamaño, por lo que no están optimizados para la compresión de textos de tamaño pequeño. Por otro lado, los algoritmos que están construidos para comprimir textos cortos son en mayoría de licencia privada, por lo que no están disponibles para el publico general y los que son de licencia abierta son escasos.

Por consecuencia, los algoritmo existentes de código abierto y privado que son capaz de comprimir textos de tamaño pequeño son escasos, y los compresores tradicionales, que son los usados comúnmente, no tienen un comportamiento exitoso al ejecutarse en textos cortos.

Capítulo 2

Estado del Arte y Conceptos Fundamentales

2.1. Definición de Tasa de Compresión

En esta memoria se usará la tasa de compresión de un texto como métrica comparativa de los diversos métodos y algoritmos de compresión que se expondrán y evaluarán.

Por consecuencia, se define la tasa de compresión como la razón entre el tamaño de un texto comprimido y el tamaño del texto sin comprimir. Esta razón es expuesta en la Ecuación 2.1:

$$\frac{texto_comprimido}{texto_sin_comprimir} \tag{2.1}$$

Por lo tanto, se entiende que cuando la tasa de compresión es inferior a 1 (ó un 100 %), hay éxito en la compresión, puesto a que el texto resultante del algoritmo de compresión es más pequeño que el texto sin comprimir. Por contraparte, cuando la tasa de compresión es mayor a 1, significa que la compresión no redujo el tamaño del texto, ya que, el texto comprimido es más grande que el texto original.

2.2. Compresores de Texto Corto

Desde finales del siglo XX se han realizado diversos estudios para encontrar un algoritmo que pueda comprimir de manera eficiente textos de tamaño corto. Una algoritmo recientemente hallado, fue publicado en el año 2006, el cual, expone un método para compresión de textos cortos basado en tablas hash[9].

El algoritmo se construyó para ser utilizado por dispositivos móviles, por ende, su complejidad es baja, debido a que los recursos disponibles de los dispositivos móviles son acotados y además se debe ocupar el mínimo de batería para que sea usado en los aparatos mencionados.

El método de compresión se basa en la codificación y decodificación mediante el uso de una tabla hash que convierte las entradas a binario. Esta codificación es revisada a través de bits de paridad para evitar la pérdida de información. Como la solución descrita está realizada para dispositivos acotados, el algoritmo utilizado está limitado a los recursos de dichos dispositivos, por consecuencia, se privilegia más la portabilidad que la tasa de compresión. Sin embargo, se observa que cuando se utilizan segmentos de memoria pequeños (32 bits), la tasa de compresión llega a un 20 %, que es un valor muy bueno para una compresión de texto, pero sólo en casos excepcionales se puede obtener esta tasa. La Figura 2.1, muestra las tasas de compresión para el algoritmo LCC, en función de un dataset de prueba que incluye textos cortos como versículos de la biblia.

En Julio del 2008 se presenta el paper *Compression of small text files*[7], en donde se sugiere un nuevo algoritmo (BWT) para archivos de texto de tamaño pequeño usando la trasformación de Burrow-Wheeler. En este documento se explica una adecuación del algoritmo Quine-McCluskey que es usado para la minimización de funciones de Booleanos.

El algoritmo mencionado en el párrafo anterior funciona de la siguiente manera, primero el texto de entrada es procesado y convertido en bytes (octetos) mediante el algoritmo de

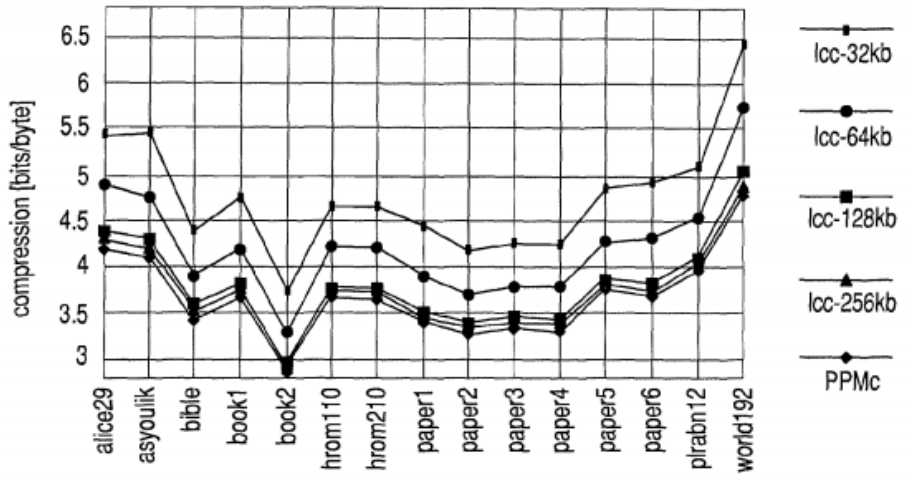


Figura 2.1: Tasa de compresión de Algoritmo LCC, para textos cortos pre definidos.
Fuente: Low Complexity Compression of Short Messages[9]

Quine-McCluskey, luego se generan vectores de la entrada convertida a bytes (octetos), los que son mapeados con una tabla de mapeo, para luego obtener un texto simplificado que es comprimido con el algoritmo de Burrow-Wheeler.

Este algoritmo se probó comparándolo con los compresores más usuales, incluido Huffman, en donde se obtuvieron resultados favorables, que en los mejores casos, la tasa de compresión es un 50 % y en promedio la tasa de compresión del algoritmo es de 80 %.

A pesar de que un 80 % es una tasa alta, éste es mejor otros algoritmos de compresión. El Cuadro 2.1 muestra el desempeño de BWT versus el desempeño de otros algoritmos, incluido algoritmo de Huffman.

En 2012 se publica el algoritmo *b64pack*[3], que tiene como función ser un método rápido de compresión de textos cortos sin pérdidas (principalmente para la compresión de mensajes SMS de teléfonos celulares). El algoritmo funciona mediante tres procedimientos base, una de transcodificación de la información, una segunda de sustitución de palabras mediante un diccionario, y una última de compresión sin pérdidas. Un ejemplo de la codificación usada se encuentra en la Figura 2.2.

Table 2
Comparison between base algorithm and algorithms with BWT

	sms.txt (93,288 B)		*bible.txt* (4017,008 B)	
	CS (bytes)	CR (%)	CS (bytes)	CR (%)
MinHuff	108,696	11.652	4,447,367	11.071
MinBwt	73,941	79.26	2,946,782	73.36
MinBwtL	96,786	10.375	3,891,008	96.86
MinBwtC	88,934	95.33	3,590,441	89.38
MinBwtS	81,881	87.77	3,260,340	81.16

Table 4
Comparison with standard algorithms

	sms.txt (93,288 B)		*bible.txt (4,017,008 B)*	
	CS (bytes)	CR (%)	CS (bytes)	CR (%)
BWT	92,502	99.16	3,701,212	92.14
LZW	83,624	89.64	3,405,503	84.78
AHuff	78,110	83.73	3,291,873	81.95
MinHuff	108,696	11.652	4,447,367	11.071
MinBwt	73,941	79.26	2,946,782	73.36

Table 3
Comparison between base algorithm and algorithm based on BWT-enron files

	enron_nh.txt (36,891,613 B)		*enron_sh.txt (8,095,564 B)*		*enron_ssh.txt (4,533,827 B)*	
	CS (bytes)	CR (%)	CS (bytes)	CR (%)	CS (bytes)	CR (%)
MinHuff	29,548,821	80.10	7,909,173	97.70	5,018,207	11.068
MinBwt	21,055,382	57.07	6,345,648	78.38	3,706,019	81.74
MinBwtL	26,657,872	72.26	7,851,240	96.98	4,667,562	10.295
MinBwtC	25,353,586	68.72	7,470,631	92.28	4,416,945	97.42
MinBwtS	21,391,560	57.98	6,511,943	80.44	3,912,122	86.29

Cuadro 2.1: Tablas comparativas de algoritmos BWT con Huffman y LZW, con diferentes textos de tamaño corto. Fuente: Compression of small text files[7].

```
Hi□mom!□Just□wanted□to□let□you□know□that□John□moved□to
□New□York□last□Tuesday.□How□are□you?□By□the□way,□I've □
got□a□new□phone□number:□+387(87)654321.□Love□you□:-)
```

```
Hi/mom+d/Just/wanted/to/let/you/know/that/John/moved/to
/New/York/last/Tuesday+p/How/are/you+w/By/the/way+n/I+ive/
got/a/new/phone/number+r/+m387+j87+k654321+p/Love/you/+r+o+k
```

Figura 2.2: Ejemplo de transcodificación usada en algoritmo *b64pack*. Fuente: Rapid Lossless Compression[3].

B64pack está realizado bajo cuatro principios, usar requerimientos bajos de memoria, eficiencia en la compresión y descompresión, estimación eficiente de tamaño comprimido durante el proceso de compresión, e interoperabilidad entre todo tipo de computadores.

La estimación de la tasa de compresión de este algoritmo es alrededor del 75 % (valor calculado para compresión de Tweets). Este valor se logra ya que para la compresión *b64pack* no guarda información extra en el texto para la decodificación. Sin embargo, la compresión es sólo un 25 % respecto al tamaño original. Esto es porque este algoritmo se centra en la rapidez de compresión, por lo que es rápido comparándolo con compresores tradicionales

como compress, gzip y bzip2 (véase Figura 2.3).

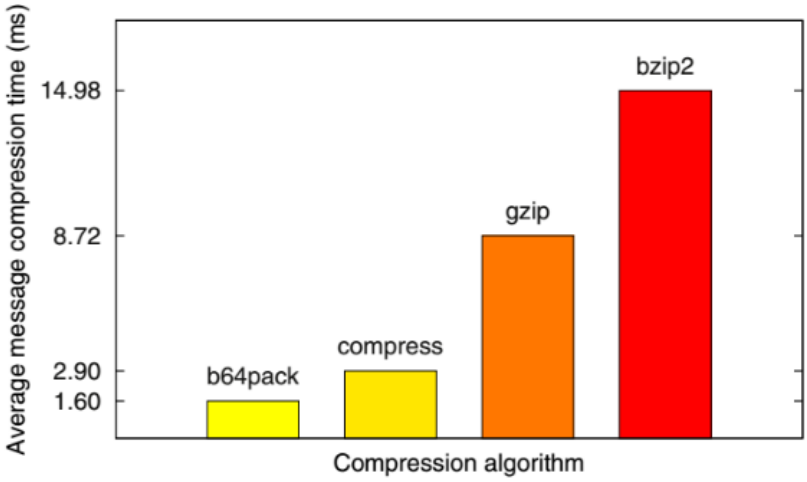


Figura 2.3: Promedio de velocidad de compresión en función de los algoritmos *b64pack*, compress, gzip y bzip2. Fuente: Rapid Lossless Compression[3].

Actualmente existen algoritmos de compresión de textos cortos usados en el mercado, como es el caso de *GSM TS03.42* [6], que es de código cerrado y es usado por compañías de telecomunicaciones. Este compresor usa Huffman como algoritmo de compresión base, con una lista de palabras fijas específica del idioma, lo que proporciona adaptabilidad. Dado a que no es posible obtener una implementación de este algoritmo, estimar el rendimiento de su tasa de compresión en tweets es difícil. (A continuación, para ser material autocontenido se incluye material de *Improving Compression of Short Messages*[6].)

SMAZ[6] es otra herramienta de compresión de mensajes cortos de código abierto. Su implementación es menor a 200 líneas de código, usando secuencias de caracteres comunes que funcionan como un diccionario. Este compresor especifica una tasa de compresión promedio de un 50 % a 70 %, siendo su mayor rendimiento cuando comprime texto en inglés.

En el Cuadro 2.2 se muestra una tabla comparativa entre el rendimiento de los algoritmos *TS03.42* y *SMAZ* al comprimir tres textos de tamaño corto, en donde se aprecia que *TS03.42* posee un rendimiento superior a *SMAZ*.

#	Uncompressed	TS03.42	SMAZ
1	358	146 (40.7%)	196 (55.7%)
2	491	249 (50.7%)	375 (76.3%)
3	601	380 (63.2%)	577 (96%)

Cuadro 2.2: Tabla comparativa de rendimiento entre TS03.42 y SMAZ. Fuente: Improving Compression of Short Messages[6].

SMAZ también es probado con mensajes de Twitter, en donde se aplicó compresión a un corpus de tweets recopilados entre octubre y noviembre de 2012, con un muestreo aleatorio con variedades de idiomas y formas. En el Cuadro 2.3 se muestra los resultados de la experimentación con tasas de compresión resultantes altas, en donde, el promedio de compresión es de $81,4\,\%$, con lo que se concluye que el diccionario utilizado por *SMAZ* no es lo suficientemente robusto para el tipo de mensajes que posee Twitter.

Corpus	Type	Messages	Avg. Length	SMAZ
Pie Floater*	English	28	171	60.6%
SMAZ README file	English	71	44	64.1%
British English SMS Corpus [28]	SMS	450	80	64.8%
SMS SPAM Collection v.1 [27]	SMS	5,547	82	71.6%
Private Twitter Corpus	Twitter	348,994	54	81.4%

*The text from http://en.wikipedia.org/wiki/Pie_floater.

Cuadro 2.3: Tabla de rendimiento de *SMAZ*, en donde figura la compresión de tweets. Fuente: Improving Compression of Short Messages[6].

Una variación del algoritmo *BWT* mencionado en un comienzo es el algoritmo *ShortBWT*, que adquiere las propiedades de la transformación Burrows-Wheeler para textos cortos. En contraste con *SMAZ* y *GSM TS03.42*, este algoritmo no utiliza un diccionario pre existente,

sino que utiliza estadísticas pre calculadas en función de frecuencia de caracteres.

Como *ShortBWT* no está disponible públicamente, su desempeño es examinado con muestras predefinidas y se estima que puede comprimir cadenas de 200 caracteres a un 55 %.

El 2013 se presenta el algoritmo *Stats3*[6], cuyo enfoque es modelar estadísticas representativas de mensajes de Twitter y usarlas como un esquema de compresión no adaptativo, con una codificación aritmética Back-End.

Este algoritmo se centra en probabilidades de sintaxis de un conjunto de mensajes de Twitter, en donde se juega con la probabilidad de los 69 caracteres más comunes, tratando todas las letras como minúsculas. Además, se utiliza la probabilidad de que la primera letra de una palabra sea mayúscula o minúscula, basándose en palabras anteriores y por último usa la probabilidad de que la longitud del tweet sea entre 0 y 1024 caracteres, que es la longitud máxima que soporta *stats3*.

El algoritmo en cuestión usa un diccionario generado, en donde las palabras con frecuencia menor a 5 se descartan y este diccionario se hace mediante a modelos predictivos que ayudan a la compresión del texto.

El Cuadro 2.4 muestra los resultados de rendimiento (tasas de compresión) de stats3 versus los algoritmos de compresión *SMAZ* y *TS03.42*, en donde se observa un rendimiento de stats3 muy alto en relación al algoritmo *SMAZ*.

Además de los algoritmos ya expuestos existen también esquemas comerciales, patentados y de código cerrado, en donde se observan ofertas de rendimiento entre 60 % y 70 % [2] y entre un 45 % y 50 % [13] en los mejores casos.

	Messages	TS03.42	SMAZ	stats3
GSM sample message 1	1	40.7%	55.7%	32.8%
GSM sample message 2	1	50.7%	76.3%	43.8%
GSM sample message 3	1	63.2%	96%	48.8%
SMAZ sample message set	17	-	64.1%	49.6%
Pie Floater	28	-	60.6%	40.6%
SMAZ README file	71	-	64.1%	45.6%
British English SMS Corpus	450	-	64.8%	41.4%
SMS SPAM Collection v.1	5,547	-	71.6%	44.1%
Private Twitter Corpus	145,333	-	80.7%	46.64%

Cuadro 2.4: Tabla de rendimiento de *stats3*, contrarrestado con *SMAZ* y *TS03.42*. Fuente: Improving Compression of Short Messages[6].

2.3. Algoritmo de Huffman

Existen muchos algoritmos de compresión en la literatura, uno de los más clásicos es el algoritmo de compresión de Huffman (desarrollado por D. Huffman en 1952), que usa una codificación basada en la frecuencia de símbolos que ocurren en el texto a comprimir.

El algoritmo mencionado opera mediante un mecanismo de codificación leyendo un árbol binario de frecuencias creado a partir del archivo a comprimir, el cual, entrega un código binario asociado a cada carácter en el texto, con el que se codifica y decodifica el archivo y/o fichero.

Para comprimir y/o descomprimir un archivo, existen una serie de pasos a realizar cuando se utiliza el algoritmo de Huffman, los que se detallan a continuación:

1. Creación de una lista enlazada que almacene la frecuencia de cada carácter presente en el texto y/o archivo a comprimir.
2. Ordenar la lista de menor a mayor, según la frecuencia.
3. Transformación de cada elemento de la lista en un árbol.
4. Fusión de todos los árboles en un solo árbol: En este paso se eligen los dos primeros arboles de la lista y se fusionan, sumando las frecuencias de cada rama. El árbol resultante se integra a la lista en el orden correspondiente, para luego fusionar los arboles mas pequeños nuevamente. Esto se repite hasta que solo quede un solo árbol.
5. Se asigna un código binario a cada carácter. Esto se realiza buscando dicho carácter en las hojas del árbol, en donde el camino para llegar a la hoja será el código binario.
6. Finalmente se codifica el archivo usando los códigos obtenidos a partir del árbol binario.

Cabe destacar que para poder descomprimir el archivo es necesario guardar el árbol que generó la codificación para luego poder decodificarlo, ya que este árbol de frecuencia es único y sin él, no se podría volver a obtener el archivo original. En base a lo anterior, se anexa al archivo comprimido el árbol de binario de frecuencias que proporciona la información necesaria para la descompresión.

Un ejemplo del uso del algoritmo se muestra a continuación[1] :
Si se tiene el texto *"ata la jaca a la estaca"*, primero se cuenta la frecuencia de cada carácter y se ingresa a un lista.

$$\text{`}\,\text{'}(5) \longrightarrow a(9) \longrightarrow c(2) \longrightarrow e(1) \longrightarrow j(1) \longrightarrow l(2) \longrightarrow s(1) \longrightarrow t(2)$$

Luego, se ordena la lista de caracteres según la frecuencia, en forma ascendente:

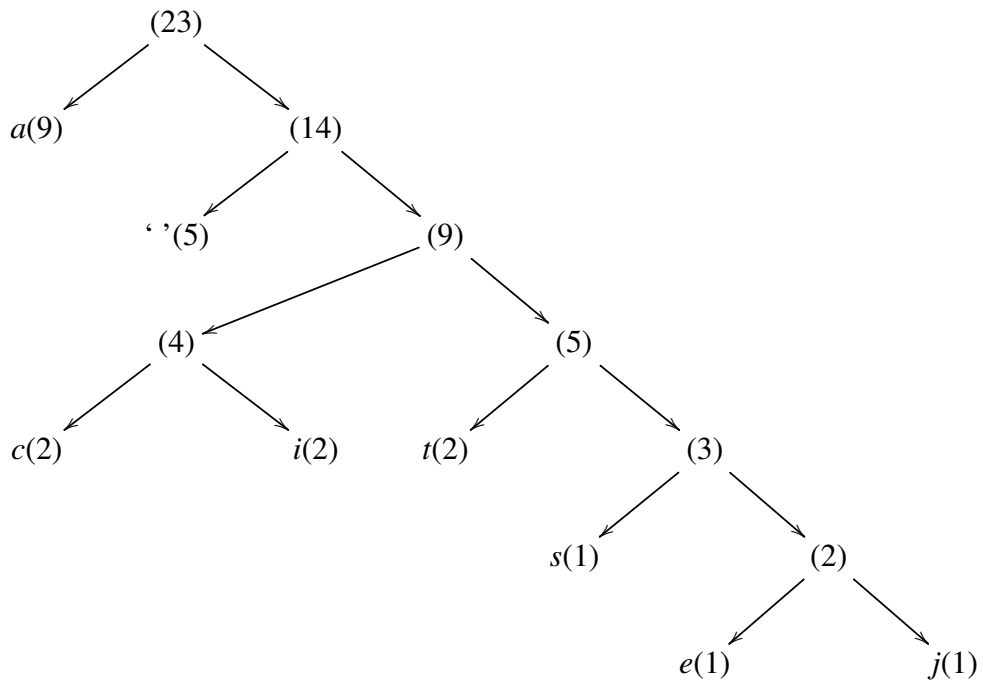
$$e(1) \longrightarrow j(1) \longrightarrow s(1) \longrightarrow c(2) \longrightarrow l(2) \longrightarrow t(2) \longrightarrow \text{`}\,\text{'}(5) \longrightarrow a(9)$$

Cada elemento de la lista se trata como un árbol, en donde se fusionan los árboles con menor frecuencia (están primero en la lista), para luego insertar el árbol resultante en la lista, manteniendo el orden de la misma. Si se fusionan los dos primero arboles de lista anterior, el resultado es el siguiente:

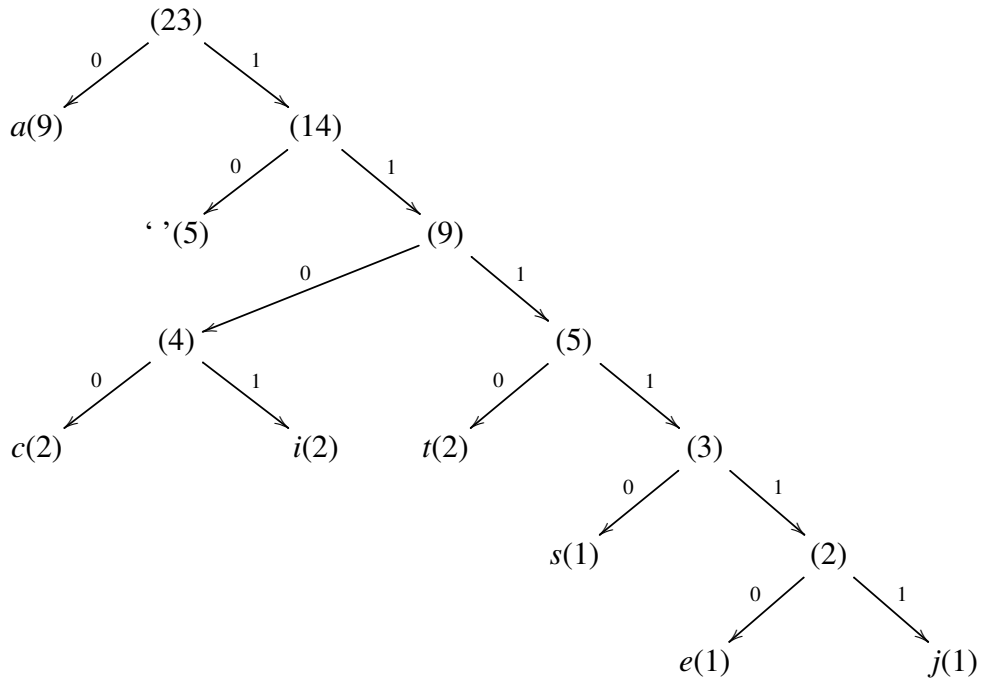
$$\begin{tikzcd}
s(1) \arrow[r] & (2) \arrow[r] \arrow[dl] \arrow[dr] & c(2) \arrow[r] & l(2) \arrow[r] & t(2) \arrow[r] & \text{`}\,\text{'}(5) \arrow[r] & a(9) \\
e(1) & & j(1) & & & &
\end{tikzcd}$$

El proceso anterior se realiza en forma iterativa hasta que quede un sólo árbol en la lista. El resultado de esto, basado en el texto de ejemplo es el siguiente árbol de frecuencias.

[1] ejemplo completo: `http://articulos.conclase.net/?tema=algoritmos&art=huffman&pag=000`



Para poder codificar y comprimir, a cada rama del árbol, que es parte del camino para llegar a las hojas (símbolos), se le asigna un cero si es una rama izquierda y un uno, si es una rama derecha (esta regla se aplica de forma arbitraria). Con esto, el árbol de codificación resultante es el siguiente:



En base al árbol binario construido, se procede a comprimir codificando el texto deseado. Para ello, cada carácter o símbolo se reemplazará por un el código binario generado por el árbol, por ejemplo, el carácter *a* será reemplazado por un *0*, ya que para llegar a la hoja que contiene dicho carácter, se recorren las ramas del árbol comenzando desde la raíz, formando el código binario *0*. El Cuadro 2.5 posee la codificación binaria para todos los símbolos del ejemplo dado.

Símbolo	**a**	**‘ ’**	**c**	**l**	**t**	**s**	**e**	**j**
Codificación	0	10	1100	1101	1110	11110	111110	111111

Cuadro 2.5: Codificación binaria de Huffman para símbolos de texto *“ata la jaca a la estaca”*

Ahora, con la codificación del Cuadro 2.5, se codifica todo el texto *“ata la jaca a la estaca”*, con lo que se obtiene la codificación en binario mostrada en el Cuadro 2.6.

a	t	a	‘ ’	l	a	‘ ’	j	a	c	a	‘ ’
0	1110	0	10	1101	0	10	111111	0	1100	0	10

a	‘ ’	l	a	‘ ’	e	s	t	a	c	a
0	10	1101	0	10	111110	11110	1110	0	1100	0

Cuadro 2.6: Codificación binaria de Huffman para texto *“ata la jaca a la estaca”*.

Terminados todos los pasos anteriores, la codificación y compresión de Huffman esta lista. Sólo queda guardar todos los bits de unos y ceros como bytes. Si se observa el texto *“ata la jaca a la estaca”*, posee 23 caracteres que son traducidos en 23 bytes. La codificación de Huffman tiene un total de 60 ceros y unos (bits), que si expresan en bytes, la compresión logra un peso de 8 bytes.

2.4. Otros Compresores Tradicionales

Hoy en día existen compresores que tienen un uso muy popular en los computadores. Por ejemplo, GNU Zip (Gzip) es un compresor libre de GNU usado en plataformas linux que se atribuye generalmente a archivos con extensión .tar.gz o .tgz, publicado en 1992. Su implementación es una combinación entre algoritmos convencionales de codificación y la codificación de Huffman.

También los compresores de la línea *7-Zip* son usados en la actualidad. El algoritmo Lempel–Ziv–Markov chain algorithm (LZMA) creado en 1998, que, fue diseñado para proporcionar alta compresión, descompresión rápida y bajos requerimientos de memoria[10]. La compresión está basada en un diccionario, que logra una tasa de compresión más alta de gzip generalmente.

LZMA posee una variante llamada LZHAM (LZ, Huffman, Arithmetic, Markov) que se enfoca en obtener un mayor rendimiento en la descompresión de ficheros muy grandes, utilizando la codificación de Huffman en el algoritmo.

Capítulo 3

Compresión de tweets usando algoritmo de Huffman con modelos predefinidos de compresión

3.1. Propuesta

Esta memoria propone un método de compresión para textos cortos mediante el algoritmo de Huffman, que es un algoritmo clásico de compresión de textos, usando una modificación para mejorar el desempeño al comprimir textos pequeños.

¿Por qué usar el algoritmo de Huffman y no otro compresor clásico? Anteriormente se dijo que algunos compresores clásicos como GZip y LZMA ocupan dentro de sus algoritmos la codificación de Huffman, este es uno de los motivos principales, ya que, si se optimiza uno de los algoritmos base, los algoritmos que ocupan la base modificada también aumentarán su rendimiento.

Por otro lado, si se compara el rendimiento de Huffman, Gzip y LZMA cuando se ejecutan

sobre textos pequeños, Huffman tiene el peor desempeño. En la Figura 3.1 se muestra el porcentaje de compresión de tweets cuando se comprime con algoritmos clásicos sin efectuar alguna modificación en ellos.

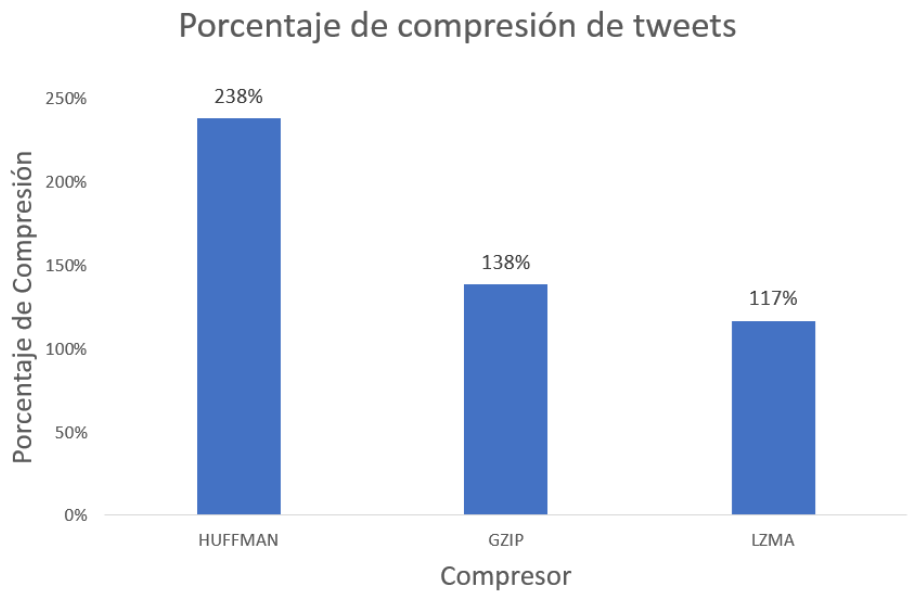


Figura 3.1: Gráfico comparativo de rendimiento en la compresión de textos cortos de compresores clásicos. Fuente: Elaboración propia.

El hecho de que Huffman posea un peor desempeño que sus pares lo hace ser un candidato especial para probar una modificación que lo haga ser eficiente al momento de comprimir textos pequeños.

En la Figura anterior (3.1) se muestra la compresión aplicada a tweets individuales (textos pequeños), lo que no es óptimo para un compresor clásico. Esta premisa se cumple en la Figura 3.2 que muestra la compresión de tweets pero ahora agrupados en conjuntos de 1.000, 5.000, 25.000, 50.000, 100.000 y 500.000 tweets, en donde, a medida que mayor es el tamaño del conjunto, más grande son los archivos a comprimir.

La Figura 3.2 expresa claramente pendientes decrecientes a medida que aumenta el tamaño

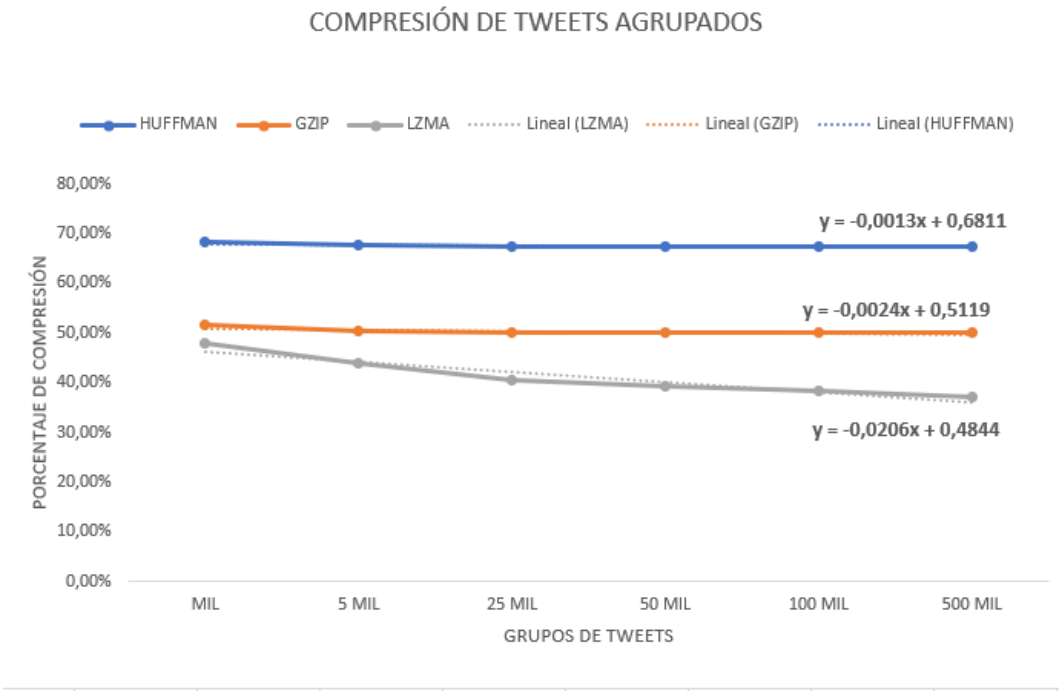


Figura 3.2: Gráfico comparativo de rendimiento en la compresión de tweets en grupos por diferentes compresores clásicos. Fuente: Elaboración propia.

del documento a comprimir (o a medida que crece el tamaño de la agrupación). Este aspecto refuerza la idea que entre más pequeño es un documento, un compresor clásico posee un peor rendimiento al momento de codificar y/o comprimir.

Todo lo señalado hace notar la importancia de poder modificar el funcionamiento normal de un algoritmo clásico para que al comprimir un texto pequeño pueda hacerlo de manera eficiente, cumpliendo el propósito principal de un compresor: reducir el tamaño de un archivo, y Huffman es un candidato particular por su alta tasa de compresión (cercana al 70 % en textos grandes) y su modularidad, que hace que otros compresores incluyan la codificación de Huffman en sus procesos intermedios de compresión.

Con lo anterior señalado, se pretende implementar una modificación al algoritmo de Huffman para que este sea capaz de comprimir textos cortos de manera eficiente, mejorando el rendimiento del algoritmo al usarse en su forma tradicional.

Para comprimir textos, Huffman utiliza una tabla de frecuencia de símbolos, la que es usada para codificar y decodificar el texto comprimido. Por lo tanto, para cada compresión y descompresión, es necesario acceder a la tabla mencionada, lo que implica que ella sea anexada al texto comprimido. Esta situación es ineficiente al hablar de textos cortos, ya que dicha tabla en general es más grande que el mismo texto.

La propuesta para optimizar el algoritmo de Huffman para textos cortos, es no almacenar de la tabla de frecuencias del texto comprimido. Esta propuesta tiene un efecto directo en la tasa de compresión, dado que en textos pequeños se reduce considerablemente el tamaño del texto comprimido y, por lo tanto, la tasa de compresión disminuye.

La metodología descrita, utilizada en textos grandes, carece de sentido, puesto que anexar una tabla de frecuencias al texto no tiene mayor incidencia en el tamaño de éste. Sin embargo, como se dijo anteriormente, la extracción de esta tabla de frecuencia de símbolos toma sentido cuando se contextualiza la compresión a un texto corto.

Con todo lo señalado, se propone usar las tablas de frecuencias más representativas de un conjunto determinado de documentos, las que se usarán para codificar y decodificar cada archivo. Se pretende obtener 256 diferentes tablas de frecuencias, ya que con esto se puede utilizar un único *byte* de referencia para identificar el modelo usado para codificar un texto. Las 256 tablas de frecuencias que se obtendrán se denominarán *modelos predefinidos de compresión*.

Para efectuar las pruebas de compresión usando la modificación propuesta, se usará un conjunto de datos de tweets, que son las publicaciones de la red social Twitter. Estas publicaciones se escogieron ya que son textos cortos, de no más de 140 caracteres (280 desde noviembre de 2017). Se segmentará el conjunto total de tweets en dos, asignando un conjunto de pruebas para realizar las evaluaciones de cada método, y un conjunto de entrenamiento

para crear los modelos predefinidos de compresión.

3.2. Implementación

La implementación del compresor de Huffman fue realizada en el lenguaje de programación *C* en la universidad de Alicante en España [8], por el profesor Prof. F. Javier Gil Chica. Dicho código fue modificado con el objetivo de adaptar la implementación a la necesidad de comprimir textos cortos.

La implementación del algoritmo se basa en la construcción de una lista enlazada, que en cada nodo representa un carácter del alfabeto (256 caracteres ASCII en total), la frecuencia del carácter en el texto y la codificación binaria del mismo. El ordenamiento de la lista enlazada creada se realiza mediante recursión, para que luego se cree la codificación binaria de Huffman en cada nodo.

Cabe destacar que el archivo comprimido por el algoritmo de Huffman guarda en la tabla de frecuencia de símbolos sólo los caracteres presentes en el texto, por lo tanto, la tabla de frecuencias será tan grande como la cantidad de caracteres diferentes que ocurran en el texto.

Al aplicar el algoritmo de Huffman en su forma tradicional en textos cortos se obtiene una situación particular, puesto a que el texto comprimido es más grande que el textos sin comprimir. Para revertir esta situación se suprimió del texto comprimido la tabla de frecuencias. Este método no implica que ya no se use una tabla de frecuencias para comprimir o descomprimir un texto, sino que ahora se tendrán tablas de frecuencias genéricas que son modelos predefinidos de compresión, las que se referenciarán mediante un byte anexado al documento codificado y comprimido.

La supresión de tablas de frecuencias propuesta, se realiza de la siguiente manera: antes de comprimir se procesa una muestra del conjunto de datos de mensajes cortos (conjunto de entrenamiento), con el objetivo de obtener los modelos de tablas de frecuencias más representativos de la muestra.

Al comprimir se elegirá el modelo pre definido que más se ajuste al tweet a procesar, y mediante este modelo se comprimirá y descomprimirá el texto.

En consecuencia, estos son los dos principales problemas que afronta esta memoria:

1. Decidir que modelos predefinidos almacenar, es decir, encontrar los modelos genéricos que se usarán para comprimir.
2. Decidir desde los modelos ya almacenados, cuál usar para comprimir un texto dado.

En esta memoria se indicarán dos maneras de elegir los modelos de compresión más representativos de un conjunto de textos cortos: elección de modelos de manera estocástica y elección de modelos mediante clustering.

3.2.1. Modelos Estocásticos

La elección de los modelos predefinidos de manera estocástica se realiza mediante la elección aleatoria de subconjuntos de tweets de la muestra, es decir, se generan 256 subconjuntos de documentos, los que están conformados por un número definido de textos elegidos de manera aleatoria.

Cada subconjunto de tweets tiene como propósito generar un modelo predefinido de compresión. Para ello, a cada uno se le creará una tabla de frecuencias general, que tendrá las frecuencias de todos los símbolos de los documentos que estén en él.

La tabla de frecuencias general se crea mediante el procesamiento de cada texto del subconjunto, en donde, se obtienen las frecuencias de símbolos individuales que son sumadas a la tabla de frecuencias general.

Cada tabla de frecuencias general construida es por sí sola un modelo predefinido de compresión, ya que representa la distribución de frecuencias de todo un subconjunto y partir de ella se obtiene un árbol binario de Huffman, con el que puede codificar y decodificar un texto.

La Figura 3.3 muestra la elección de tweets de manera aleatoria, sin importar la forma ni propiedades de ellos. En el ejemplo, a partir del conjunto principal se generan dos subconjuntos sin discriminar el color de los elementos. Cada subconjunto creado es la base para crear un modelo predefinido de compresión.

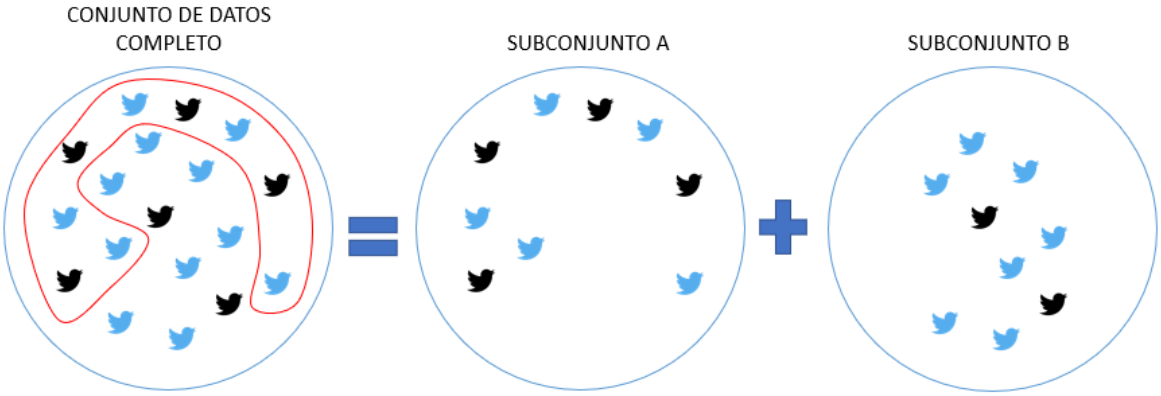


Figura 3.3: Diagrama explicativo de elección de modelos estocásticos para la compresión de tweets.

3.2.2. Modelos mediante Clustering

"Clustering" es el agrupamiento de una serie de datos con respecto a un criterio particular (generalmente se usa distancia o similitud). La idea de esto es poder agrupar tweets similares y mediante ellos generar modelos predefinidos de compresión.

El algoritmo de clustering usado en esta ocasión es el *Algoritmo de CLARA (Clustering Large Applications)*[1], que es una implementación del algoritmo de clustering PAM (Partitions around medoids) sobre un subconjunto de datos, creado múltiples muestras del conjunto de datos y aplicando PAM sobre cada una de ellas. En otras palabras, la búsqueda de Clusters se realiza mediante la elección de k representantes de un conjunto de datos, en donde se genera el agrupamiento según la distancia promedio entre los objetos y el representante k de manera iterativa.

Para que el algoritmo de Clara funcione, es necesario generar una matriz de distancia de distribuciones de frecuencias. Para ello, se comparan las distribuciones de frecuencias a través de la distancia o divergencia de Jensen-Shannon[5], que es la medida de distancia simétrica de la divergencia KL [4] que se explicará más adelante.

La divergencia de Jensen-Shannon se define como:

$$D_{JS}(P\|Q) = \frac{1}{2}D_{KL}(P\|M) + \frac{1}{2}D_{KL}(Q\|M)$$

$$\text{donde, } M = \tfrac{1}{2}(P + Q)$$

Con la matriz de distancia creada, el proceso de clusters generará 256 subconjuntos de textos, que, al igual que en el método estocástico, se procesarán los textos de cada subconjunto, generando una tabla de frecuencia general, la que será un modelo predefinido de compresión.

La Figura 3.4 muestra un agrupamiento de documentos por una característica particular, en este caso, agruparon por color los documentos. Por consecuencia, a partir del conjunto original se generan dos subconjuntos, en donde cada uno de ellos contienen documentos semejantes (en el diagrama, elementos del mismo color quedan en el mismo subconjunto).

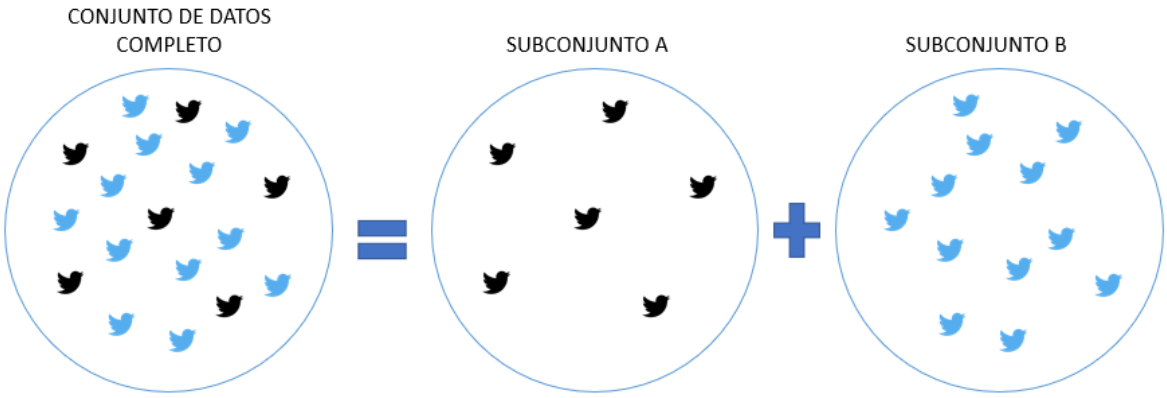


Figura 3.4: Diagrama explicativo de elección de modelos mediante clustering para la compresión de tweets.

3.2.3. Cotejo entre modelo predefinidos de compresión y tweet

Luego de obtener los modelos predefinidos de compresión, es necesario encontrar qué modelo se adecua de mejor manera al tweet a comprimir. La forma en que se realiza esta comparación es mediante el cotejo de las tablas de frecuencias.

Cada tabla de frecuencia es también una distribución de probabilidad de cada símbolo, por lo que la comparación se traduce al cotejo entre distribuciones de probabilidad de símbolos. La distancia de Kullback - Leibler[4] es una medida no simétrica de similitud o diferencia de distribuciones, en donde, dadas las distribuciones de probabilidad P y Q, se define la divergencia KL para variables discretas como:

$$D_{KL}(P||Q) = \sum_{i=1} P(i) ln \left(\frac{P(i)}{Q(i)} \right)$$

Por consecuencia, cuando la divergencia es cercana a cero, indica que las distribuciones de frecuencia de símbolos son cercanas, y por contraparte, cuando la distancia crece, las distribuciones de frecuencias de símbolos no son similares.

Entonces, para encontrar el modelo predefinido de compresión que mejor se ajuste a un texto dado, se calcularán la divergencias entre el texto y todos los modelos predefinidos de compresión, en donde, se elegirá el modelo predefinido de compresión que posea la divergencia más cercana a cero.

Capítulo 4

Resultados

Todas las pruebas realizadas se hicieron en un computador con sistema operativo *Ubuntu 16.04.3 LTS*, con memoria RAM de tamaño 48 GB y un total de 12 núcleos virtuales (6 físicos) *Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz*.

4.1. Descarga de conjunto de datos para pruebas

Antes de comenzar a realizar pruebas del algoritmo de compresión, es necesario disponer de un conjunto de datos suficiente para proceder a las pruebas.

Como el objeto de estudio será la red social Twitter, el conjunto de datos son los tweets descargados desde la api oficial de Twitter. Para ello, se escogieron tweets georreferenciados desde Inglaterra, ya que los textos provenientes de este país son en inglés, lo que permite la comparación con las pruebas de otros compresores.

Cabe destacar que para el estudio se considera sólo el texto del tweet y no los datos extras que son propios del mismo, como el usuario que lo creó, fecha, hora, localidad, etc.

Los tweets descargados para la construcción del conjunto de datos fueron adquiridos en un lapso de tiempo comprendido entre el 5 Julio y 5 Noviembre de 2017, en donde, se recaudaron 1.007.237 tweets georreferenciados. De estos, 201.568 (20 % del total de tweets descargados) fueron utilizados para entrenar los modelos predefinidos de compresión y el resto, fueron utilizados para la realización de las pruebas de compresión.

La Figura 4.1 muestra el gráfico de la distribución de los tweets descargados en función de su tamaño medido en Bytes. En él, se observa que el grueso de tweets obtenidos se concentra entre 1 bytes y 160 bytes aproximadamente. Cabe señalar que, el tamaño promedio de los tweets pertenecientes al conjunto de datos de pruebas es de $86,6$ bytes, en donde, los tweets con mayor tamaño (mayor a 300 bytes) tienen una frecuencia muy baja en comparación a la gran mayoría de los datos.

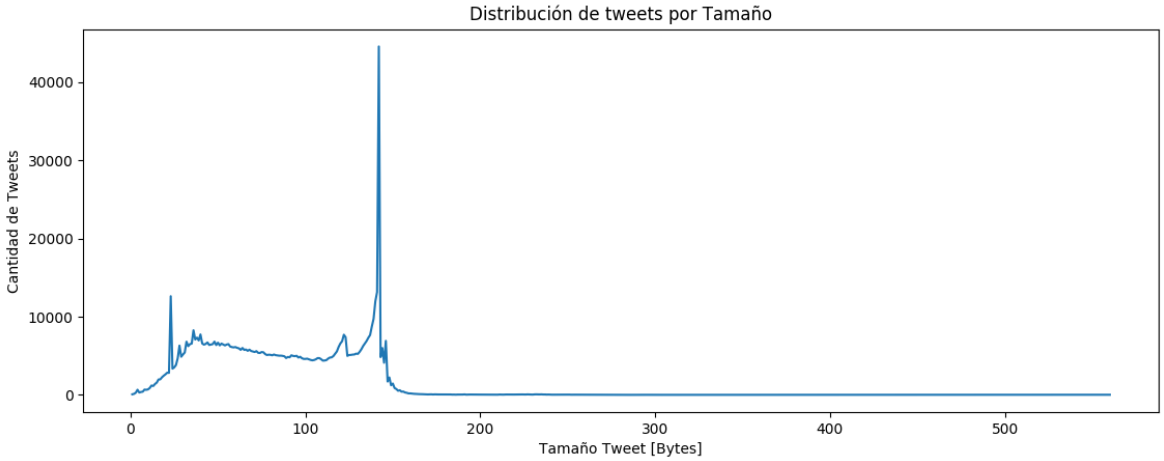


Figura 4.1: Distribución de tweets en función de su tamaño en disco.

4.2. Creación de modelos predefinidos de compresión

El método estocástico propuesto, genera 256 grupos de 10.000 tweets seleccionados de manera aleatoria. En donde, para cada grupo de tweets se obtiene las frecuencias de los caracteres, y con ello se construye el modelo pre definido de compresión.

La generación de la selección aleatoria de tweets toma un tiempo de $2,306$ segundos y la creación de los modelos predefinidos de compresión demoró un tiempo de $1679,04$ segundos. Por lo que el tiempo para la selección de tweets y la creación de los modelos predefinidos de compresión es de $1681,346$ segundos (aproximadamente 28 minutos) y el tamaño total que ocupan estos modelos es de 320 KB.

A diferencia de los modelos estocásticos, los modelos generados por Clusters requieren un tratamiento previo, ya que es necesario crear los Clusters y diferentes archivos temporales que hacen posible la clasificación de tweets.

El primer paso es la generación de una matriz de distancias, que relaciona y compara las distribuciones de frecuencias entre todos los tweets de la muestra. En esta oportunidad, no se usarán los 201.568 tweets del conjunto de entrenamiento, ya que el tiempo de computo crece en forma cuadrática.

Se realizaron pruebas que fallaron por la incapacidad de computo de la máquina usada, en las que usaron matrices de (100.000×100.000) y (50.000×50.000), llegando a una matriz que si se pudo computar de (25.000×25.000) tweets.

La nueva matriz de distancias de dimensión 25.000×25.000, tuvo un tiempo de procesamiento de 4 horas con 38 minutos y 43 segundos. Su espacio en disco abarca $5,3$ GB de almacenamiento. Esta matriz, si puede ser procesada con la memoria disponible en la máquina, con lo que el algoritmo de clustering Clara demoró un tiempo ejecución de 26.071

segundos (o 7 horas, 14 minutos y 31 segundos), ocupando un 99 % de CPU.

Los archivos resultantes del algoritmo de Clara permiten la creación de dos tipos de modelos predefinidos de compresión. El primero es usando el documento más representativo de cada subconjunto y el segundo es usando todos los textos de cada subconjunto.

Para el primer tipo (tweet más representativo de cada subconjunto resultante), cada tweet representante proveerá una distribución de frecuencias para construir un modelo de compresión. En el segundo tipo, los modelos predefinidos son construidos en base a la suma de todas las frecuencias de símbolos de los documentos de cada subconjunto, por lo que se tendrá una frecuencia general para cada subconjunto procedente del algoritmo de Clara.

Los modelos predefinidos creados a través de los tweets más representativos de los conjuntos obtenidos mediante el algoritmo de Clara, fueron procesados en $1,45$ segundos y los modelos predefinidos creados a partir de todos los componentes de los clusters, se gestaron en un tiempo de $121,93$ segundos.

El Cuadro 4.1 muestra la comparación entre los procedimientos para crear los modelos predefinidos de compresión. En él se observa que los modelos predefinidos de compresión realizados con el método estocástico abarcan un mayor conjunto de tweets que al realizar un entrenamiento por clusters. Si observa el tiempo de creación de índices y árboles, la suma de ellos es mucho menor cuando se usa el procedimiento estocástico. Esto se debe a que cuando se usan clusters es necesario procesar los datos muchas veces para poder agrupar el conjunto de datos y poder realizar la selección de los documentos más parecidos.

Es importante destacar, que, los modelos pre definidos de compresión fueron creados a partir de los primeros tweets descargados (que son más pequeños en tamaño que los tweets del conjunto de pruebas, ya que en noviembre de 2017, Twitter aumentó el máximo de caracteres permitidos en cada comentario), las pruebas y análisis se enfocarán en los tweets inferiores

Modelo	Tamaño muestra (tweets)	Tiempo Creación índice [s]	Tiempo Creación modelos	Tamaño de modelos [KB]
Estocástico	201.568	2,306	1679,04	320
Tweets Representativos de Clusters	25.000	42.794	1,45	320
Clusters Completos	25.000	42.794	121,93	320

Cuadro 4.1: Comparación de los procesos de creación de los modelos predefinidos de compresión.

a 300 bytes.

4.3. Compresión de Tweets

El conjunto de datos usados para experimentar con los algoritmos de compresión es 805.669 tweets procedentes de Inglaterra, que ocupan un espacio en disco de $66,54$ MB. Cabe destacar que existen archivos considerados outliers y son los tweets con tamaño superior a 300 bytes (342 tweets de la muestra). Estos archivos están contenidos en el estudio, sin embargo, en algunos gráficos no se visualizan con el objetivo de dar mayor claridad a los resultados.

Dicho lo anterior, para obtener una comparación entre los tamaños de archivos se comprimieron los tweets del conjunto de datos con algoritmo de Huffman tradicional, es decir, a cada tweet comprimido se le incorporó la tabla de frecuencia de símbolos que se usa para codificar y decodificar la información contenida en el archivo.

El proceso de compresión realizado sobre el total de tweets tardó un tiempo de 2 horas, 24 minutos y 40 segundos, ocupando un 10 % de CPU y un espacio de $158,37$ MB de almacenamiento en disco, que es un tamaño ampliamente superior al del conjunto de datos sin comprimir, esto se refleja en que la tasa de compresión es de 238 %.

Según lo hablado en los párrafos anteriores, Huffman comprime usando la frecuencia de ocurrencia de símbolos en el texto, lo que indica una compresión personalizada para cada documento. Este hecho supone que la mayor tasa de compresión es cuando se codifica usando la tabla de frecuencias del mismo tweet, sin adjuntar dicha tabla al documento codificado. Por lo que se define la *compresión optima* de Huffman como la compresión típica de Huffman sin anexar la tabla de símbolos (Se entiende que esta compresión es referencial, ya que, si se comprime y no se guarda la tabla de frecuencia, no se podrá descomprimir en el futuro).

El proceso de compresión óptimo tardó 2 horas con 25 minutos y 32 segundos, ocupando un 10 % de procesamiento en CPU y un espacio de $38,22$ MB, lo que señala una tasa de compresión de 57 %.

Ya creados los modelos predefinidos de compresión, el proceso de compresión tradicional cambia, ya que se inserta un paso extra, el cotejo entre las distribuciones de frecuencia de símbolos, con el fin de obtener el modelo predefinido de compresión que más se ajuste a un tweet dado.

Para la compresión usando modelos predefinidos de manera estocástica, el tiempo usado para comprimir los 805.669 tweets fue de 2 horas, 42 minutos y 25 segundos, usando un 76 % de CPU y espacio en disco de $48,94$ MB, tamaño de compresión inferior al original, lo que indica una tasa de compresión de 74 % en promedio.

Y, usando el método de clusters se crearon dos tipos de modelos predefinidos de compresión, uno con los 256 tweets representantes (uno de cada subconjunto de datos) y el otro con los 256 subconjuntos diferenciados de la muestra (usando todos los tweets de cada subconjunto).

Usando los modelos pre definidos de compresión, gestados a partir de un *único representante* de cada subconjunto diferenciado, la compresión del conjunto de datos de pruebas tardó 15.240 segundos (4 horas con 14 minutos), en donde los textos comprimidos ocuparon un

espacio en disco de 135, 65 MB, lo que indica una taza de compresión promedio de 204 %.

Por otro lado, la compresión usando la totalidad de los textos de cada subconjunto diferenciado de la muestra (*clusters* completos) tardó 9.534 segundos (alrededor de 2 horas y 39 minutos), en donde los 805.669 tweets comprimidos, ocuparon un espacio en disco de 48, 39 MB, lo que denota una tasa de compresión promedio de 73 %.

La Figura 4.2 muestra un resumen comparativo de las tasas de compresión medias de cada método usado para la compresión de tweets. Como se ha señalado, la compresión *óptima* de Huffman no es usable en la práctica, ya que esta codificación no se puede decodificar, pero es una métrica que ayuda a obtener una cota inferior al momento de comparar. Esta cota se marca en la figura siendo la que tiene menor tasa de compresión. Cabe destacar que los métodos inferiores a un 100 %, son métodos que si comprimen en la práctica, por consecuencia, aquellos métodos que están sobre esta tasa de compresión (Huffman y representante único), en vez de comprimir, el archivo crece de tamaño.

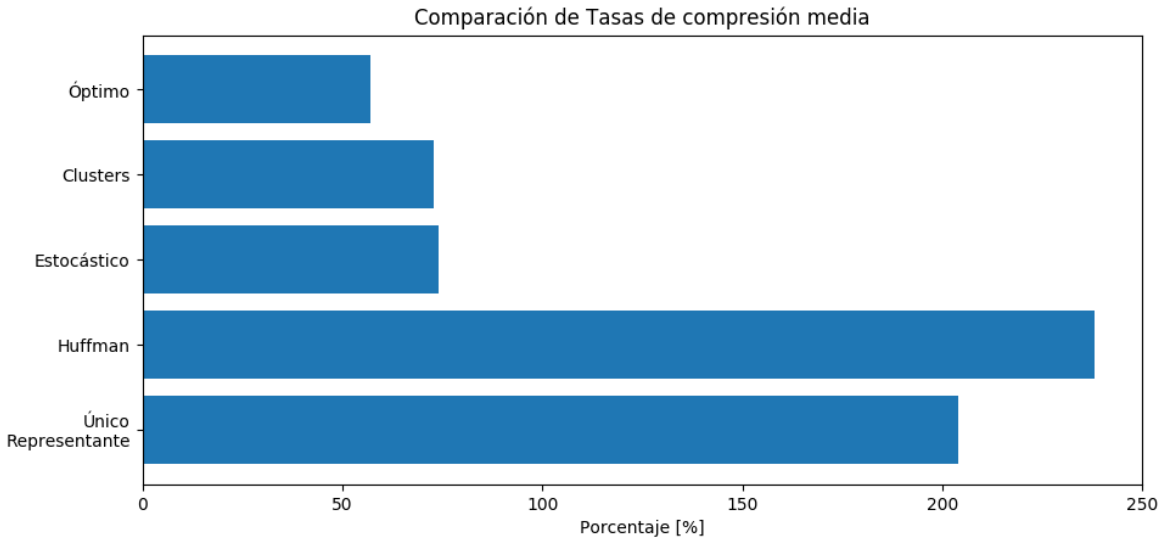


Figura 4.2: Gráfico comparativo de las tasas de compresión media.

Si sólo se analizan las compresiones medias observables en la Figura 4.2, se diría empíricamente que el mejor método aplicable en la práctica en el de Clusters y el peor es el algoritmo de Huffman en su forma pura. Pero, dar una conclusión de este tipo es apresurado ya que es necesario obtener más información.

La Figura 4.3 muestra los tamaños promedios en disco de los tweets luego de haber aplicado los algoritmos de compresión sobre ellos. Para la realización de la figura se agruparon los tweets de acuerdo al tamaño original, en conjuntos por rango de 50 bytes. La figura en cuestión considera tweets hasta 300 bytes.

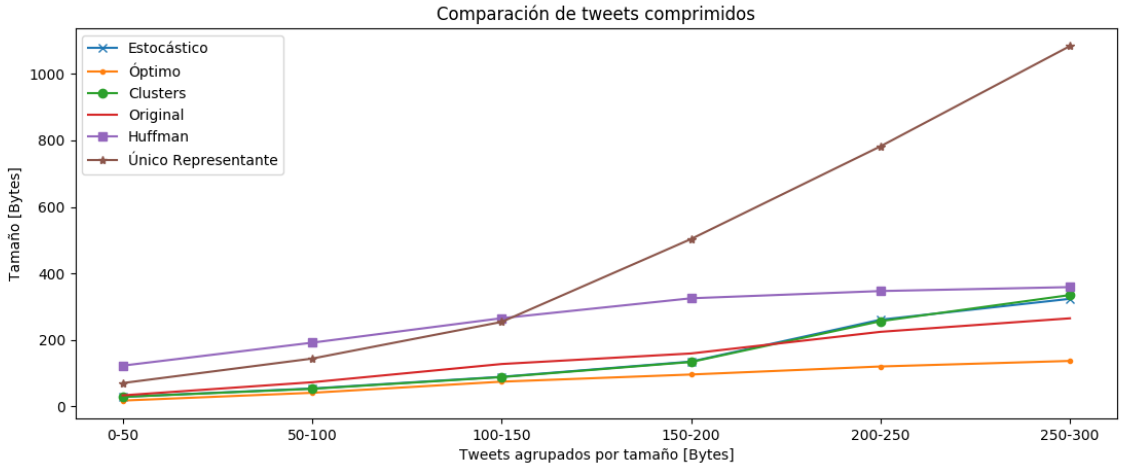


Figura 4.3: Gráfico comparativo entre tamaños promedios de compresiones de tweets al usar diferentes métodos de compresión.

La línea roja (o también la linea lisa) presente en la Figura 4.3 indica el tamaño original de un tweet en el disco, por consiguiente, cualquier curva sobre ella indica que el algoritmo de compresión tiene como resultado un archivo de mayor tamaño al original y las curvas por debajo de ella, indican que el algoritmo logró disminuir el tamaño del archivo, comprimiéndolo.

También es posible observar que los mejores resultados son cuando se ocupan clusters y el método estocástico para construir los modelos predefinidos de compresión. Esta situación ocurre hasta el umbral de tweets de 200 bytes, luego de esto la compresión es ineficiente.

Por otro lado, la curva de único representante, denota un comportamiento ascendente en la compresión, lo que indica que este método no es eficiente al momento de comprimir. Cabe destacar que existen casos (no reflejados en la Figura 4.3) en donde la compresión por único representante si es eficiente, y esto sucede cuando el modelo es muy similar al tweet a comprimir.

Si se agrega al análisis el gráfico presente en la Figura 4.4, en donde se manifiestan las tasas de compresión de los mismos tweets expuestos en la Figura 4.3, se ratifica que los algoritmos que mejor comprimen el conjunto de datos de textos diminutos son cuando se utilizan modelos predefinidos *estocásticos* y *clusters*; y por otro lado se confirma el comportamiento no eficiente cuando se comprime usando como modelos predefinidos de compresión los *únicos representantes* de cada cluster.

Cabe señalar que las líneas que están sobre la curva *Original* (linea roja y lisa) en la figura 4.4, son resultados de compresiones en donde el tamaño promedio de los archivos resultantes es más grande que el tamaño promedio de los archivos originales luego de aplicar el algoritmo de compresión, y por consecuencia, la tasa de compresión media es mayor a 1 o mayor al 100 %.

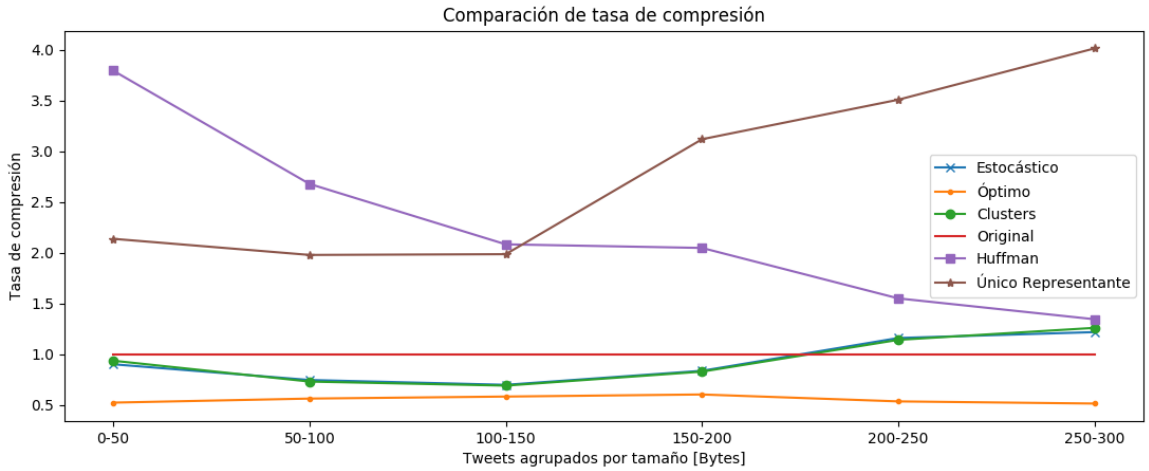


Figura 4.4: Gráfico comparativo entre tasas promedios de compresión de tweets al usar diferentes métodos de compresión.

Se observa también en la figura 4.4 que la mejor tasa de compresión ocurre en los archivos entre 100 y 150 bytes.

Para ver en más detalle los resultados con mejor desempeño se creó la Figura 4.5 que incluye sólo aquellas curvas que están bajo la línea *original*, es decir, aquellas que su compresión tiene tamaño inferior al archivo original y por consecuencia, su tasa de compresión es menor a 1.

Se observa que las curvas presentes en los gráficos de la figura 4.5 son cuatro, en donde la que posee mejor rendimiento es la *óptima*, que como se explicaba anteriormente, su ejecución no es posible. Las otras tres, son las curvas de compresión en base modelos predefinidos por *clusters*, *estocásticos* y la curva que representa el tamaño original y/o tasa de compresión 1. Estas curvas poseen un comportamiento muy similar e incluso sus desempeños tienden a igualarse en algunos puntos de la curva, siendo en ocasiones mejor la compresión mediante clusters.

También, se observa más claramente que el mejor desempeño de Huffman con modelos pre definidos de compresión es cuando se usa en archivos con tamaño entre 100 bytes y 150 bytes, en donde existen tasas de compresión de 70 % (modelos estocásticos) y 69 % (modelos mediante clusters).

Ya que los métodos que tienen éxito en la compresión poseen un comportamiento similar en el grueso de tweets, es decir, las tasas de compresión al momento de su ejecución son parecidas, señalar cúal de los dos es mejor conlleva un análisis más profundo que la simple comparación entre la compresión del mismo tweet. Por consiguiente, se presenta el Cuadro 4.2 que indica una comparación entre los dos mejores métodos de compresión de textos cortos encontrados de manera empírica. En él, se observa que los rasgos más diferenciadores de ambos métodos son el tamaño del conjunto de entrenamiento y el tiempo de entrenamiento.

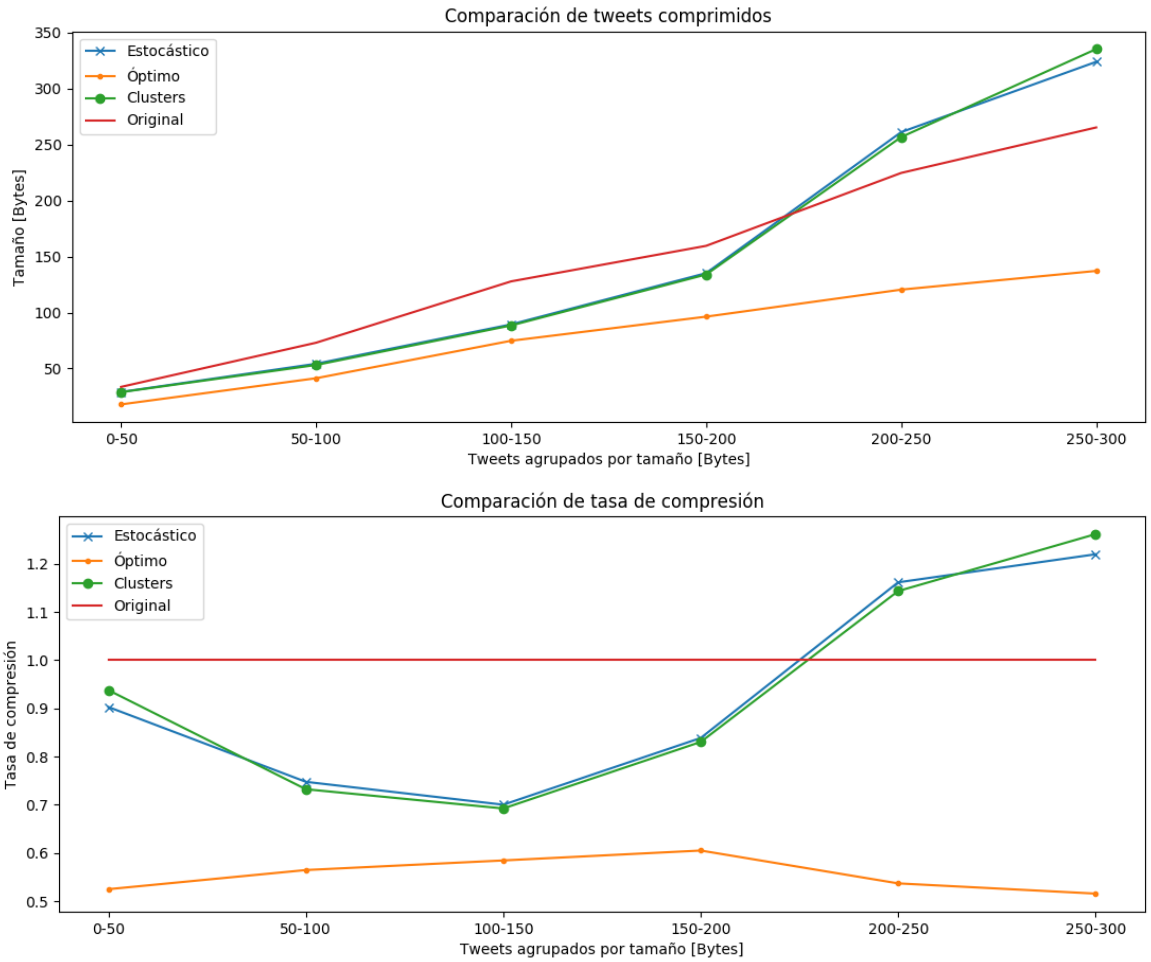


Figura 4.5: Gráficos comparativos de desempeño de compresiones con resultados positivos.

Método	Tamaño conjunto de entrenamiento	Tiempo de Entrenamiento [s]	Tiempo de Compresión[s]	Tasa de compresión media
Estocástico	201.568	1.681,35	9.745	74 %
Clusters	25.000	42.915,93	9.534	73 %

Cuadro 4.2: Tabla comparativa de rendimiento de los dos métodos que empíricamente mostraron resultados positivos.

Para elegir algún método en específico para comprimir de los dos mejores encontrados hay que sopesar los datos que se tienen para entrenar, dado a que, si se tiene muchos datos de entrenamiento con relación al conjunto total de datos, conviene usar el método estocástico, dado a que el tiempo de entrenamiento es menor. Sin embargo, si los datos de prueba son acotados con relación al conjunto de datos total, sin duda conviene usar el método basado en Clusters.

Dicho lo anterior, cabe señalar que, a pesar de que las tasas de compresión fueron similares, el algoritmo de Huffman con modelos predefinidos de compresión basados en clusters usó un conjunto de entrenamiento que es el $2,48$ % del total y obtuvo tasas de compresión promedio mejores que el método estocástico, que se entrenó con el 20 % de la muestra. Con esto, cabe señalar, que, si se tienen los recursos físicos y temporales para entrenar los modelos de compresión, es mejor usar clusters para obtener los modelos predefinidos de compresión.

4.4. Huffman con modelos predefinidos de compresión v/s SMAZ

En la sección del estado del arte de este trabajo, se mencionó un algoritmo creado para comprimir textos cortos de código abierto. Este algoritmo se llama Smaz y tiene por particularidad usar un diccionario para comprimir y en su implementación tener menos de 200 líneas de código.

Smaz en el paper *improving compression of short messages*[6] se explaya una tasa de compresión en un corpus privado de tweets de un $81,4\,\%$ (véase figura 2.3), sin embargo, se probará el algoritmo Smaz implementado por *Salvatore Sanfilippo*[11](implementación oficial) sobre un conjunto de tweets propios, con el fin de obtener una tasa de compresión empírica del algoritmo y realizar comparaciones de tasas de compresión sobre un mismo conjunto de datos.

Al probar la implementación de Smaz[11] se encontró problemas al comprimir algunos documentos, esto se debe a que las referencias a personas en Twitter se realizan mediante un carácter especial que Smaz lo reconoce como un término de mensaje, por lo que no comprime la cadena completa. También existe otras combinaciones de símbolos que Smaz comprime de manera inadecuada, por esta razón, se seleccionaron de la muestra de tweets sólo los que Smaz comprime bien y con ellos se realizaron las pruebas.

Por consecuencia, para las pruebas de compresión se usaron 216.041 documentos cortos, que fueron comprimidos con los algoritmos Smaz y Huffman con modelos predefinidos de compresión (estocástico y clusters).

Al comprimir el total de documentos, Smaz obtuvo una taza de compresión media de $78,426\,\%$, Huffman con modelos estocásticos obtuvo una tasa de compresión de $75,632\,\%$ y Huffman con modelos mediante clusters $75,629\,\%$. Por lo que la modificación del algoritmo de Huffman de modelos predefinidos de compresión posee un mejor rendimiento que el algoritmo Smaz al comparar sólo tasas de compresión.

En la Figura 4.6 se muestra un gráfico comparativo de los tamaños promedio de compresión de los tres algoritmos en discusión en función del tamaño del archivo original. En la figura se observa que a pesar de que Huffman con modelos predefinidos posee una mejor tasa de compresión media, desde archivos mayores a 150 bytes Huffman obtiene un peor desempeño que Smaz y en archivos menores a 150 bytes Huffman con modelos predefinidos de compresión

supera a Smaz.

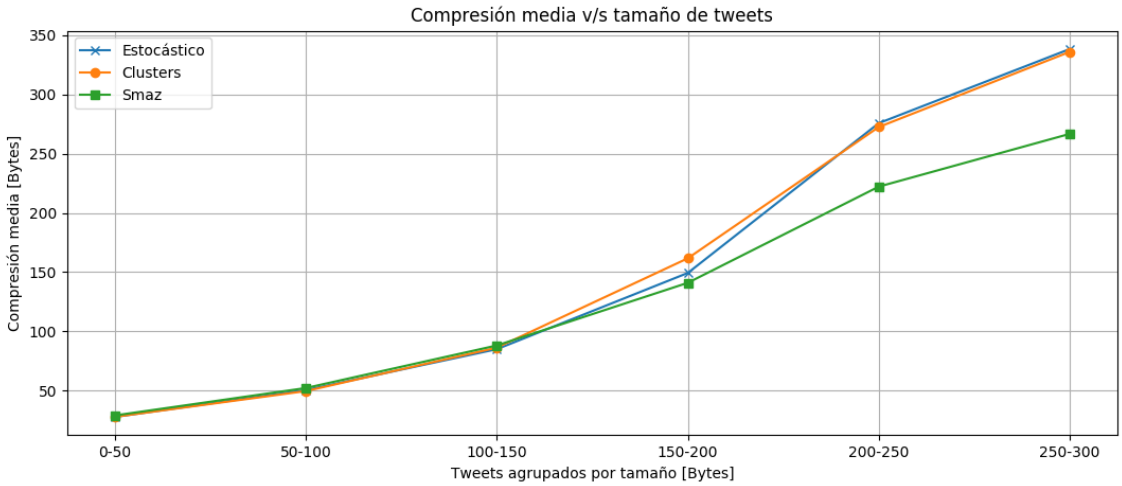


Figura 4.6: Gráfico comparativo de tamaño de compresión media en función del tamaño del tweet sin comprimir, incluyendo el algoritmo de compresión Smaz.

La figura 4.7 muestra el tamaño de compresión media de los algoritmos involucrados para tweets con tamaño menor que 150 bytes.

Smaz y Huffman con modelos predefinidos poseen un comportamiento muy similar en textos menores a 150 bytes. En este tramo Huffman con modelos predefinidos de compresión estocástico y mediante clusters tienen un mejor rendimiento que Smaz, sin embargo, luego de pasar la cota de los 150 bytes, Smaz tiene un mejor rendimiento al comprimir.

En general, Huffman con modelos pre definidos de compresión tiene mejor tasa de compresión, esto se debe a que la gran mayoría del conjunto de pruebas fluctúa en textos de tamaño 140 bytes, y en estos tipos de textos, Huffman posee un mejor rendimiento.

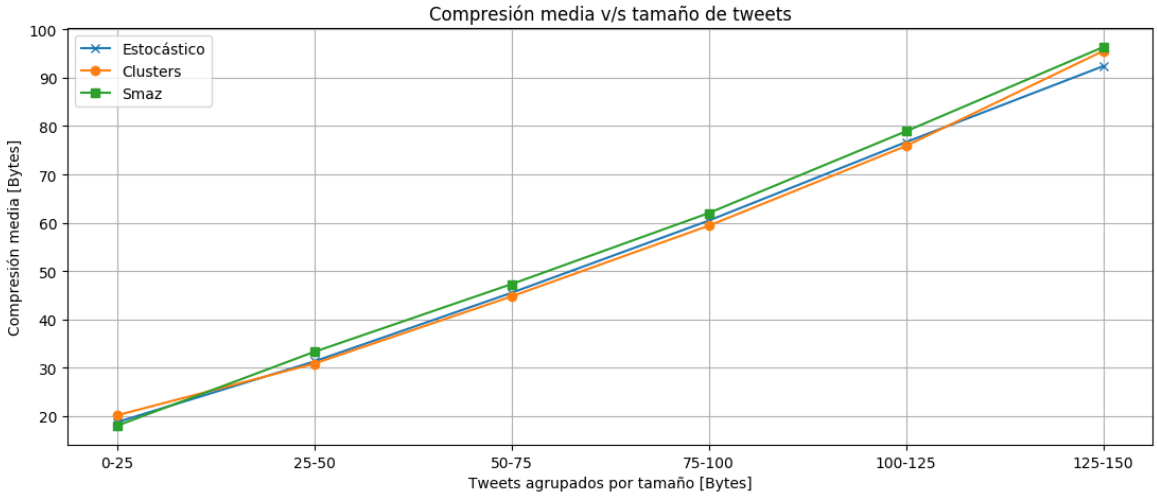


Figura 4.7: Gráfico comparativo de tamaño de compresión media en función del tamaño del tweet sin comprimir, incluyendo el algoritmo de compresión Smaz, para textos menores que 150 bytes.

Capítulo 5

Conclusiones y Trabajo A Futuro

Los modelos predefinidos de compresión tuvieron éxito al comprimir textos cortos (en este estudio se utilizaron tweets). Al usar la compresión de Huffman en su forma tradicional, la tasa de compresión promedio fue de 238 %, es decir, el texto crece más del doble de su tamaño al codificar. Por contraparte, al usar modelos predefinidos de compresión se logró una tasa de compresión media que realmente achica el tamaño del archivo original.

Usando el método de clusters se obtuvieron los mejores resultados, en donde la tasa compresión media de la data de tweets fue de 73 %, esto es usando sólo el $2,48$ % de tweets de la muestra para realizar el entrenamiento del algoritmo de agrupamiento. Sin embargo, la compresión es eficiente sólo para tweets menores a 150 bytes.

Si bien, los modelos predefinidos de compresión obtuvieron una tasa de compresión media muy similar, estos se entrenaron con un 20 % del conjunto de datos original, que es una cifra muy mayor al conjunto de entrenamiento que ocupó el otro método mencionado, pero la compresión es estable en todo el espectro de datos del conjunto pruebas.

Por lo tanto, si se tienen los recursos de tiempo y hardware para poder hacer clusters con un mayor conjunto de entrenamiento, es probable mejorar la tasa de compresión media. Sin

embargo, Huffman posee una cota inferior en la tasa de compresión media de un 57 % para textos cortos, obtenida empíricamente en este experimento, lo que indica que, pese a que se agrande el tamaño del conjunto de entrenamiento, no se podrá superar esta tasa de compresión.

Usando Huffman con modelos predefinidos de compresión, la tasa de compresión en textos cortos (73 %) se acerca a la tasa de compresión de Huffman obtenida empíricamente al aplicar el algoritmo en textos grandes (entre 65 % y 70 %) - *Véase Figura 3.2* -. Por otro lado, la modificación hecha al algoritmo de Huffman logra acercarse a las tasas de compresión media de algoritmos diseñados para comprimir textos cortos.

También, se probó empíricamente que Huffman con modelos pre definidos de compresión (ambos, clusters y estocásticos) posee una mejor tasa de compresión que Smaz (algoritmo de compresión construido para textos pequeños), cuando los archivos a comprimir son inferiores a 150 bytes. Luego de este tamaño, Smaz tiene una mejor tasa de compresión.

Otro aspecto importante a destacar tiene relación con la calidad del conjunto de entrenamiento, ya que el conjunto usado para entrenar los modelos predefinidos de compresión poseía textos de a lo mas 140 caracteres, por consecuencia, las compresiones más eficientes fueron cuando los textos a codificar tenían un tamaño similar. Por el contrario, cuando los textos eran más grandes (situación desencadenada por el aumento del tamaño de los comentarios de Twitter en noviembre de 2017), las compresiones no fueron eficientes.

Estudiar el comportamiento de la variación de la tasa de compresión media al aumentar el tamaño del conjunto de entrenamiento usando modelos predefinidos de compresión por "Cluster" es sin duda un trabajo deseable a futuro, ya que en esta memoria de titulo sólo se experimentó con una muestra de entrenamiento reducida por limitaciones de hardware.

Por otro lado, este documento únicamente abarcó el estudio de compresiones en un sólo

idioma: inglés, creando modelos predefinidos de compresión para dicho lenguaje y aplicando estos modelos a textos con el mismo idioma. Por ello, es deseable el estudio del comportamiento de los modelos predefinidos aplicados a diferentes idiomas, por ejemplo, usar modelos predefinidos de textos en inglés para comprimir textos en español, alemán, portugués y otro, con el objeto de encontrar similitudes y diferencias entre los modelos de diferentes dialectos.

Bibliografía

[1] Clara (clustering large applications). En LING LIU and M. TAMER ÖZSU, editors, *Encyclopedia of Database Systems*, Boston, MA. Springer US, 2009, pp 330–330.

[2] Paul Gardner-Stephen and Swapna Palaniswamy. Serval mesh software-wifi multi model management. En :*Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, New York, NY, USA. ACM, 2011, ACWR '11, pp 71–77.

[3] Kenan Kalajdzic, Samaher Hussein Ali, and Ahmed Patel. Rapid lossless compression of short text messages. *Computer Standards & Interfaces*, 37:53 – 59, 2015.

[4] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[5] M. L. Menéndez, J. A. Pardo, L. Pardo, and M. C. Pardo. The jensen-shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 1997.

[6] R. Challans J. Hampton J. Lakeman P. Gardner-Stephen, A. Bettison and C. Wallis. Improving compression of short messages. *International Journal of Communications, Network and System Sciences*, 6(12):497–504, 2013.

[7] Jan Platoš, Václav Snášel, and Eyas El-Qawasmeh. Compression of small text files. *Adv. Eng. Inform.*, 22(3):410–417, jul 2008.

[8] España Prof. F. Javier Gil Chica, Universidad de Alicante. Algoritmo de huffman [en línea] ⟨`http://www.dfists.ua.es/~gil/huffman.c`⟩. [consulta: 15 Noviembre 2017].

[9] S. Rein, C. Guhmann, and F. H. P. Fitzek. Low-complexity compression of short messages. En :*Data Compression Conference (DCC'06)*, March 2006, pp 123–132.

[10] David Salomon. *Data Compression: The Complete Reference*. 2007. With contributions by Giovanni Motta and David Bryant.

[11] Salvatore Sanfilippo. Smaz - compression for very small strings [en línea] ⟨https://github.com/antirez/smaz⟩. [consulta: 27 Diciembre 2017].

[12] David Sayce. Number of tweets per day? [en línea] ⟨https://www.dsayce.com/social-media/tweets-day/⟩. [consulta: 8 Septiembre 2017].

[13] Acticom GmbH SMSzipper. Smszipper [en línea] ⟨http://smszipper.com⟩. [consulta: noviembre 2016].