UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Departamento de Ciencia de la Computación

TRABAJO DE DIPLOMA

Título: Detección de spam utilizando clasificadores incrementales

Autor: Asiel Díaz Benítez

Tutores: Ing. Alberto Verdecia Cabrera

Dra. Leticia Arco García

Santa Clara, Junio 2018

Me gustaría dedicar este trabajo a toda mi querida familia ...

AGRADECIMIENTOS

Me gustaría agradecer a mi familia por todo el apoyo que me han brindado y por hacerme la persona que soy, en especial a mi querido abuelo por haber sido un padre para mí, a todos mis profesores por todo el conocimiento que me regalaron durante estos 5 años. También me gustaría agradecer a mi novia y a mi suegra por apoyarme en cada momento y a mis tutores por todo su apoyo y paciencia.

SÍNTESIS

El correo electrónico, pese a su antigüedad, sigue siendo una forma popular de comunicación por ser muy económico y fácil de usar. La mayoría de los correos electrónicos que son enviados en la actualidad son correos spam los cuales suponen una pérdida de tiempo y dinero para usuarios y empresas. Para contrarrestar el spam se han creado herramientas que intentan detectarlo de forma automática. No obstante, no existe un método que permita la detección de spam con una precisión absoluta dado que los spammers cambian la forma de los spam para confundir a los filtros anti-spam. Sin embargo, en el área del aprendizaje automático existen algoritmos capaces de reconocer estos patrones (conocido como cambio de concepto) y adaptarse al mismo. El presente trabajo se propuso como objetivo detectar de manera eficiente los correos spam utilizando clasificadores incrementales, de los cuales algunos son capaces de adaptarse al cambio de concepto, para lo cual se diseñó un modelo para la detección personalizada de spam y se realizaron experimentos para evaluar el desempeño de los algoritmos a utilizar en el modelo, obteniendo resultados favorables que permitieron llegar a la conclusión de que este tipo de clasificadores son eficaces para la detección de spam, con un elevado grado de precisión en las predicciones, destacándose el clasificador FASE, el cual fue seleccionado para formar parte de la herramienta final propuesta.

ABSTRACT

Email, despite its age, still is a popular form of communication because it is very economic and easy to use. Most of the emails that are sent today are spam emails, which represent a waste of time and money for users and companies. To counteract spam, tools have been created that try to detect it automatically. However, there is no method that allows the detection of spam with absolute precision since spammers change the form of spam to confuse anti-spam filters. However, in the machine learning area there are algorithms capable of recognizing these patterns (also know as concept drift) and adapting to them. The objective of the present work is to efficiently detect spam emails using online classifiers, some of which are capable of adapting to concept drift. A model for the personalized spam detection was designed and experiments were carried out to evaluate the performance of the algorithms to be used in the model, obtaining favorable results that allowed to reach the conclusion that this type of classifier are effective for spam detection, with a high percent of accuracy in the predictions, standing out from the rest FASE classifier, which was selected to be part of the proposed final tool.

ÍNDICE

FIGURAS

TABLAS

INTRODUCCIÓN

En las últimas décadas, el almacenamiento, organización y recuperación de la información se ha automatizado gracias a los sistemas de bases de datos, pero la ubicuidad de la información en formato digital ha comenzado a ser patente desde finales del siglo XX con la irrupción de Internet. Como resultado de este crecimiento, los datos en bruto se han convertido en una vasta fuente de información. Gran parte de esta información es histórica, es decir, representa transacciones o situaciones que se han producido. Aparte de su función de "memoria", la información histórica es útil para explicar el pasado, entender el presente y predecir la información futura. La mayoría de las decisiones de empresas, organizaciones e instituciones se basan en información sobre experiencias pasadas extraídas de fuentes muy diversas por lo que el verdadero valor de los datos radica en la posibilidad de extraer de ellos información útil para la toma de decisiones o la exploración y comprensión de los fenómenos que le dieron lugar . Debido a tal crecimiento y variedad de la información se hace necesario e importante el análisis de datos en ramas como: bioinformática, medicina, economía, finanzas, industria, medio ambiente, entre otras. Más importante aún es, además del conocimiento que se puede inferir y la capacidad de poder usarlo, tener un conjunto de reglas que a partir de los antecedentes, comportamiento y otras características de los datos permitan predecir su comportamiento futuro.

En la actualidad el volumen de los datos generados por sensores, Internet, dispositivos de localización, telefonía y muchos otros, está en constante aumento. El tamaño de estos datos es potencialmente infinito, debido a su constante generación y así, es necesario procesarlos con recursos limitados de cómputo. Para este procesamiento es factible el uso de técnicas de aprendizaje automático. El aprendizaje automático se puede clasificar en dos tipos, dependiendo de cómo se presenten los ejemplos de entrenamiento (Nishida, 2008): aprendizaje por lotes y aprendizaje incremental. En el aprendizaje por lotes, los datos deben ser recolectados y almacenados antes de ser procesados, por lo que su uso no es factible para procesar datos generados constantemente en el tiempo. Sin embargo, el aprendizaje incremental permite procesar flujos de datos de manera secuencial y los aprende uno a uno. En las tareas de clasificación, un flujo de datos es común-

mente definido como una secuencia muy grande (potencialmente infinita) de pares que se van adquiriendo a lo largo del tiempo. Estos pares, llamados instancias o ejemplos, están compuestos por un conjunto de atributos y una etiqueta de clase. Debido a la dimensión temporal de los datos (estos son adquiridos en el tiempo) y la dinámica de muchas situaciones reales, la distribución de probabilidad que regula a los datos (también llamada concepto) puede cambiar con el tiempo, un problema conocido comúnmente como cambio de concepto. Consecuentemente, los algoritmos de aprendizaje para la minería de flujos de datos deben ser actualizados con respecto a los conceptos más recientes (Gama et al., 2014).

De las diversas tareas estudiadas en los flujos de datos con cambios de concepto, la clasificación supervisada es la que ha recibido mayor atención por parte de los investigadores (Krawczyk et al., 2017). La clasificación supervisada ha sido utilizada para resolver problemas reales. Por ejemplo, en el seguimiento de objetos visuales (las cámaras de vigilancia producen un flujo de imágenes a una velocidad de muchos fotogramas por segundo) (Liu y Zhou, 2014). En la clasificación de sentimientos en los flujos de TWITTER (Bifet y Frank, 2010), ya que millones de usuarios emiten opiniones y hacen comentarios constantemente, estos mensajes fluyen de forma rápida e infinita, mientras que los vocabularios utilizados para transmitir sentimientos positivos y negativos tienden a evolucionar en diferentes contextos. Otro ejemplo es el filtrado de correos spam (Katakis et al., 2009), un usuario recibe correos electrónicos constantemente. Si un correo electrónico es spam depende del interés del usuario y dicho interés puede variar en el tiempo.

Los correos no deseados (spam) suponen actualmente la mayor parte de los mensajes electrónicos intercambiados en Internet, siendo utilizado entre otros asuntos para anunciar productos y servicios de dudosa calidad. Usualmente los mensajes indican como remitente del correo una dirección falsa. Por esta razón, no sirve de nada contestar a este tipo de mensajes. Por ahora, el servicio de correo electrónico no puede identificar los mensajes de forma que se pueda discriminar la verdadera dirección de correo electrónico del remitente, de una falsa. Esta situación que puede resultar chocante en un primer momento, es semejante por ejemplo a la que ocurre con el correo postal ordinario: nada impide poner en una carta o postal una dirección de remitente aleatoria: el correo llegará en cualquier caso. No obstante, hay tecnologías desarrolladas en esta dirección, por ejemplo: Los filtros automáticos para clasificar correos como no deseados analizan el contenido de los mensajes buscando, por ejemplo, palabras como *rolex*, *viagra* y *sex* que son las más usuales en los mensajes no deseados. Los filtros de correo electrónico utilizan algoritmos de aprendizaje automático para clasificar los correos como spam o no spam. En la literatura existen varias propuestas para detectar correos spam de forma automática, entre estas se pueden encontrar, Naive Bayes (Androutsopoulos et al., 2000; Rathod y Pattewar, 2015; Sahami et al.,

1998; Yang y Elfayoumy, 2007), máquinas de vectores de soporte (Sanghani y Kotecha, 2016) y redes neuronales artificiales (Ozawa et al., 2015).

Debido a la dimensión temporal de los correos electrónicos (se obtienen constantemente en el tiempo) la función objetivo que debe ser aprendida por los clasificadores puede cambiar en el tiempo. Esta situación conocida como cambio de concepto, complica la tarea de estimar dicha función, ya que un modelo de aprendizaje puede quedar desactualizado con respecto a los datos más recientes. Esto hace que los enfoques previos utilizados para la detección de spam sean inadecuados para la manipulación de cambios de concepto, ya que no pueden incorporar la información que se obtiene de los mensajes nuevos. De lo planteado anteriormente surge el siguiente **problema científico:**

Los spammers cambian constantemente las características de los correos; sin embargo, las soluciones que existen para la detección de spam suponen que la función de distribución que regula los datos es estática, por lo que los algoritmos clásicos para la clasificación no están en correspondencia con la dinámica de los datos, afectando la calidad de la detección ante ocurrencia de cambios.

El **objetivo general** de la investigación consiste en detectar de manera eficaz los correos spam aplicando clasificadores incrementales. Este se desglosa en los siguientes **objetivos específicos:**

1. Identificar, a partir del estudio de las principales propuestas para la detección de spam y los clasificadores incrementales, aquellos algoritmos aplicables de manera eficaz cuando existe cambio de concepto.

2. Desarrollar un modelo que permita detectar de manera eficaz los correos spam mediante la combinación de técnicas de minería de textos y de minería de flujo de datos.

3. Desarrollar un prototipo, basado en el modelo propuesto, que permita la detección personalizada de spam aplicando clasificadores incrementales.

Las **preguntas de investigación** planteadas son:

1. ¿Cuáles algoritmos de aprendizaje a partir de flujo de datos con cambio de concepto permiten una detección eficaz de los correos spam?

2. ¿Cuáles técnicas de selección de rasgos permiten representar los correos de forma tal que los algoritmos incrementales realicen una detección eficaz de spam?

3. ¿Por cuáles etapas hay que transitar para lograr la detección de spam partiendo de la representación y procesamiento textual de los correos a clasificar?

4. ¿Cómo lograr la detección personalizada de spam aplicando clasificadores incrementales?

Después de haber realizado el marco teórico se formuló la siguiente **hipótesis de investigación** como presunta respuesta a las preguntas de investigación: *La aplicación de algoritmos de aprendizaje a partir de flujo de datos con cambio de concepto en combinación con técnicas de minería de textos permite la detección eficaz de correos spam.* Para lograr los objetivos trazados y demostrar la hipótesis establecida se acometieron **las tareas de investigación** siguientes:

1. Estudio de las principales propuestas para la detección de spam.
2. Análisis de los principales clasificadores incrementales.
3. Selección de los principales clasificadores incrementales aplicables de manera eficaz cuando existe cambio de concepto.
4. Obtención de las colecciones de correos spam y no spam.
5. Estudio de las principales formas de representación textual.
6. Creación de un modelo que permita detectar de manera eficaz los correos spam mediante la combinación de técnicas de minería de textos y de minería de flujo de datos.
7. Validación del modelo propuesto en la detección de spam.
8. Implementación de un filtro, basado en el modelo propuesto, que permita la detección personalizada de spam aplicando clasificadores incrementales.

Estructura de la investigación

Capítulo 1: En este capítulo se abordan aspectos teórico-conceptuales relacionados con la detección de spam y los clasificadores incrementales. Se tratan temas referentes a los métodos de aprendizaje automatizado para la detección de spam, haciendo énfasis a las formas de representación utilizadas, cómo se aplican estos métodos, etc. En este capítulo se aborda además, la minería de flujo de datos y cuestiones relacionadas con este tema como son el cambio de concepto y los multiclasificadores incrementales.

Capítulo 2: Se presenta un nuevo modelo para la detección de spam aplicando clasificadores incrementales, mediante una descripción general del modelo y de las colecciones. Para

finalizar el capítulo, se realiza una concepción teórica-metodológica del procedimiento en general, la cual se encuentra estructurada en tres etapas con sus fases correspondientes. También se muestran los resultados experimentales.

Capítulo 3: Se presenta el prototipo para la detección personalizada de spam aplicando clasificadores incrementales. Se muestra una descripción general de la herramienta, sus módulos principales y la interfaz de usuarios.

CAPÍTULO 1

Acerca de la detección de spam y los clasificadores incrementales

En la actualidad el correo electrónico se ha convertido en una de las herramientas más importantes y utilizadas para enviar y recibir información, tanto para las empresas como para los usuario comunes. Lamentablemente al correo electrónico también se ha utilizado para enviar información que no es del interés de muchos usuarios, información que es conocida como correo basura, correo electrónico no deseado o spam.

1.1. Detección de spam

Como se mencionó, en la actualidad el correo electrónico se ha convertido en uno de los canales de comunicación más importantes tanto para las empresas como para los usuarios comunes, y es por esto que es muy utilizado para el negocio electrónico publicitario. El correo electrónico es un medio de comunicación eficiente y cada vez más popular. Al ser extremadamente económico y fácil de enviar, es también un negocio para el comercio electrónico. Spam es el término que se emplea comúnmente para designar el correo electrónico no solicitado que se envía a través de Internet (Guzella y Caminhas, 2009). La información que se envía a través de este tipo de correos generalmente es para ofrecer productos o promociones, además de contenido pornográfico y de virus informáticos. En el contenido de un mensaje spam pueden aparecer ofertas de cualquier tipo de producto, ofertas para hacerse ricos en poco tiempo, y sobre todo, contenido pornográfico. Esto puede resultar perjudicial para los usuarios, ya que sus buzones de entrada comienzan a llenarse de información que ellos no han solicitado y pierden mucho

tiempo en tratar de identificar si los correos que les llegan son legítimos o correos spam. Por esta razón, dentro del campo de la Inteligencia Artificial, se han tomado medidas para poder analizar el contenido de los mensajes aplicando técnicas de aprendizaje automático y de esta manera clasificarlos como correo legítimo o correo no deseado.

Debido a que la información en los correos electrónicos es textual, para el análisis de su contenido en la literatura se han utilizado técnicas de minería de textos para extraer información útil y novedosa a partir del cuerpo del mensaje. La representación textual es indispensable para el procesamiento del corpus del mensaje (Arco García, 2005). En esta investigación se ha seleccionado la representación espacio-vectorial (*Vector Space Model*; VSM) (Salton et al., 1975) por ser efectiva para representar documentos, ajustarse a otras formas de indexado y ser ampliamente reconocida en la comunidad de minería de textos. En VSM cada documento es identificado como un vector de rasgos en un espacio en el cual cada dimensión corresponde a términos indexados distintos (palabras). Un vector documento dado, en cada componente tiene un valor numérico para indicar su importancia. La representación textual incluye una secuencia de etapas que son indispensables para el correcto análisis del contenido del correo, a continuación se discuten cada una de estas etapas:

Transformación del corpus

El objetivo de la transformación del corpus es convertir los ficheros de entrada en una secuencia de términos lingüísticos. En el paso subsiguiente a la extracción de términos, estos *tokens* serán usados para generar rasgos significativos (índices de términos). El primer paso en la trasformación del corpus es reconocer los componentes textuales desde los diferentes formatos. Segundo, la secuencia resultante de *tokens* debe ser transformada. Posibles transformaciones son: convertir las letras todas a mayúsculas o todas a minúsculas, eliminar los signos de puntuación al final de los *tokens*, omitir los *tokens* que no contienen caracteres alfanuméricos, remplazar cada token por su raíz (lematizar), entre otros.

Reducción de la dimensionalidad

Es esencial controlar la dimensionalidad del espacio del vector documento cuando se utilizan las palabras como los términos a indexar. Las razones principales son: (i) los algoritmos de clasificación incrementales deben ejecutarse con recursos computacionales limitados y (ii) existen palabras que son irrelevantes y producen peores resultados, por tanto, eliminarlas puede aumentar la eficiencia de la clasificación a realizar. En esta investigación se utiliza el término

reducción de dimensionalidad para abarcar técnicas que controlen la dimensionalidad del vector: selección de rasgos. La eliminación de palabras de parada o gramaticales (*stop word elimination*) (Mladeni'c y Grobelnik, 1998; Yang y Pedersen, 1997), es una de las técnicas de selección utilizadas en este esquema de aplicación.

Después de pre-procesar los corpus de mensajes se procede a seleccionar los algoritmos de clasificación a utilizar. En el aprendizaje por lotes o tradicional se han utilizado varios algoritmos para la detección de spam, en la sección siguiente se discuten algunas de las propuestas anteriores.

1.2. Métodos de aprendizaje automatizado para la detección de spam

En la literatura se han propuesto varios enfoques para el filtrado de correos spam. Dentro de las técnicas para detectar spam se pueden encontrar enfoques basados en reglas, filtros colaborativos, listas blancas, listas negras, etc., siendo los algoritmos de aprendizaje automático los que se han utilizado exitosamente en este ámbito. Yang y Elfayoumy (2007) propusieron un filtro anti-spam utilizando redes neuronales y un clasificador bayesiano. Los resultados mostraron que la red neuronal obtuvo mejores resultados en cuanto a precisión.

Varios estudios de filtrado de correos spam han mostrado que Naive Bayes (NB) es efectivo para la detección de correos spam. Androutsopoulos et al. (2000) compararon el rendimiento de NB como una alternativa al enfoque de aprendizaje basado en instancias. Zorkadis et al. (2005) proponen un novedoso enfoque para el filtrado de correos spam basado en técnicas teóricas de información eficientes para la integración de clasificadores. Idris et al. (2015) propusieron una solución mejorada para la detección de correos spam mediante el reemplazo de la generación del detector aleatorio en el algoritmo de selección negativa (*Negative Selection Algorithm*; NSA) con optimización de nube de partículas (*Particle Swarm Optimization*; PSO). PSO está implementado con un factor marginal local como función de aptitud para generar detectores en un algoritmo de selección negativa.

Wijaya y Bisri (2016) proponen una combinación híbrida entre regresión logística y árbol de decisión para la detección de correos spam. La regresión logística es usada para reducir los datos o instancias ruidosos antes de ser creado el árbol de decisión. En estudios experimentales ellos utilizan el conjunto de datos Spambase del repositorio de aprendizaje automático UCI y obtienen resultados prometedores con una precisión del 91,67 %. Todos los enfoques antes mencionados

no tienen en cuenta el problema del cambio de concepto que puede presentarse en los correos electrónicos. Debido a esto, los spammers cambian activamente la naturaleza de sus mensajes para eludir los filtros anti-spam. En la siguiente sección se define formalmente el cambio de concepto.

1.3. Minería de flujo de datos

Un número creciente de situaciones del mundo real generan flujos de datos constantemente a partir de entornos no estacionarios a altas velocidades. Ejemplos de estas situaciones son los sensores, Internet, dispositivos de localización, telefonía y muchos otros. El tamaño de estos datos es potencialmente infinito, lo cual impone restricciones computacionales a los algoritmos de aprendizaje. Debido a la constante generación de los datos, no todos se pueden almacenar en memoria, por lo que los algoritmos del aprendizaje por lotes no pueden ser aplicados directamente. Los autores Ramírez-Gallego et al. (2017) enumeran las principales diferencias entre los escenarios estáticos y de flujo:

- Las instancias de entrenamiento no se tienen de antemano, si no que se obtienen de forma secuencial, generalmente una a una.
- Las instancias pueden llegar rápidamente, por lo que deben ser procesadas en tiempo real.
- El tamaño de los datos es potencialmente infinito, por lo que es imposible almacenar todos los datos entrantes en memoria.
- A cada instancia sólo se puede acceder un número limitado de veces (en casos específicos sólo una vez) y luego se descarta para limitar la memoria y el uso de espacio de almacenamiento.
- Las instancias deben procesarse dentro de un período limitado de tiempo para poder dar una respuesta en tiempo real y evitar la cola de datos.
- La función de distribución de probabilidad que genera estos datos puede variar en el tiempo. Así, los datos pasados pueden convertirse en irrelevantes o adversos con respecto a los datos actuales (cambio de concepto) (Blanco, 2014).

Varios de los puntos planteados anteriormente limitan la memoria y tiempo de procesamiento con que dispone un algoritmo para procesar estos flujos de datos. El aprendizaje incremental es

un paradigma de aprendizaje automático, donde el proceso de aprendizaje tiene lugar cada vez que surgen nuevos ejemplos y ajusta lo que se ha aprendido de acuerdo con el (los) nuevo(s) ejemplo(s). La diferencia más prominente que existe entre el aprendizaje incremental y el aprendizaje automático tradicional es que el primero no asume la disponibilidad de un entrenamiento suficiente antes del proceso de aprendizaje, sino que los ejemplos de entrenamiento aparecen con el tiempo (Geng y Smith-Miles, 2009). Por lo que este tipo de aprendizaje es capaz de incorporar la información que aporten nuevas experiencias, que antes no estaban disponibles en el conjunto de datos, al modelo que se está induciendo y capaz de hacerlo evolucionar para que cada vez represente conceptos más complejos (Blanco, 2014). El último punto señalado anteriormente ha sido ampliamente estudiado en los últimos años (Amidon, 2017; Ramírez-Gallego et al., 2017; Sethi y Kantardzic, 2017; Sun et al., 2017; Webb et al., 2017; Zhang et al., 2017). Este punto se refiere a la necesidad de adaptarse a posibles cambios en la distribución que regula los datos, que puede afectar el rendimiento del algoritmo de aprendizaje. Debido a que uno de los problemas que se tratan en este trabajo es el cambio de concepto, en la siguiente sección se describen sus características principales.

1.3.1. Cambio de concepto

La mayoría de los algoritmos de aprendizaje automático asumen que los ejemplos de entrenamiento se generan al azar, de acuerdo con una distribución de probabilidad estacionaria. Pero en entornos no estacionarios, como los flujos de datos, la distribución que genera los ejemplos puede cambiar en el tiempo.

Dentro del aprendizaje incremental (Widmer y Kubat, 1996), el problema de clasificación se define generalmente para una secuencia (posiblemente infinita) de ejemplos $S = e_1, e_2, ..., e_i...$ que se obtienen en el tiempo, normalmente uno a la vez y no necesariamente dependientes del tiempo. Cada ejemplo de entrenamiento $e_i = (\overrightarrow{x}_i, y_i)$ está formado por un vector $\overrightarrow{x_i}$ y un valor discreto y_i, llamado etiqueta y tomado de un conjunto Y denominado clase; donde cada vector $\overrightarrow{y_i} \in \overrightarrow{X}$ tiene las mismas dimensiones y cada dimensión se llama atributo.

Concepto se refiere a la función de distribución de probabilidad del problema en un punto determinado en el tiempo (Minku et al., 2010). Este concepto puede ser caracterizado por la distribución de probabilidad conjunta $P\left(\overrightarrow{X}, Y\right)$ donde $\overrightarrow{X}$ representa los atributos y Y es la clase. Por tanto existe un cambio de concepto cuando $P_{t_0}\left(\overrightarrow{X}, Y\right) \neq P_{t_1}\left(\overrightarrow{X}, Y\right)$, donde t_0 y t_1 son dos instantes de tiempo.

1.3.2. Tipos de cambio

Gama et al. (2014) distinguen dos tipos fundamentales de cambios de concepto que están presentes en muchos problemas reales:

1. *Cambios de concepto reales:* se refieren a cambios en la distribución de probabilidad a posteriori de las clases $p(Y \mid X)$. Estos cambios pueden presentarse sin que ocurran cambios en la distribución de los datos de entrada $p(X)$. Este tipo de cambio afecta directamente la frontera de decisión, que a su vez afecta el rendimiento de los algoritmos de aprendizaje. Para manipular este tipo de cambio los algoritmos de aprendizaje utilizan detectores de cambio para monitorizar alguna medida de rendimiento (por ejemplo, el error), variaciones significativas en esta medida se interpretan como cambio y consecuentemente los algoritmos realizan alguna acción para adaptarse lo más rápido posible.

2. *Cambios de concepto virtuales:* ocurren si la distribución de los datos de entrada cambia (es decir $p(X)$ cambia) sin afectar $p(Y \mid X)$. En este sentido, la distribución de los datos dentro de la misma clase cambia sin afectar la frontera de decisión. Generalmente, para manejar este tipo de cambio, muchas técnicas se centran en la distribución de los datos de entrada.

Stanley (2003) reconoce dos tipos de cambios relacionados con la frecuencia con la que se reciben las instancias que describen el nuevo concepto: abrupto y gradual. Un cambio abrupto ocurre cuando la transición entre conceptos consecutivos es instantánea. Un cambio gradual ocurre cuando el período de transición entre conceptos consecutivos contiene un cierto número de instancias de entrenamiento. Otro tipo común de cambio es el recurrente, que ocurre cuando los conceptos pueden reaparecer (Ortíz Díaz et al., 2014). Khamassi et al. (2016); Minku et al. (2010) indican que los cambios recurrentes pueden tener un comportamiento cíclico o acíclico.

- El cambio cíclico recurrente puede ocurrir según una cierta periodicidad o debido a una tendencia estacional. Por ejemplo, en el mercado de la electricidad, los precios pueden aumentar en invierno debido al aumento de la demanda, luego volver al nivel anterior en las otras temporadas.

- El cambio recurrente acíclico puede no ser periódico, es decir, no está claro cuándo el concepto puede reaparecer. Por ejemplo, los precios de la electricidad pueden aumentar repentinamente debido al aumento de los precios del petróleo y luego volver al nivel anterior cuando los precios disminuyan.

1.3.3. Algoritmos para detectar cambios de concepto

Una estrategia muy extendida para manejar cambios de concepto monitoriza constantemente alguna medida de rendimiento del modelo de aprendizaje en el tiempo. Por lo tanto, si esta medida se deteriora significativamente, se sospecha un cambio de concepto y se definen algunas acciones para actualizar el modelo de aprendizaje de acuerdo con los ejemplos más recientes.

En el campo de la minería de flujos de datos existen varias estrategias para detectar cambios de concepto. En esta estrategia los detectores de cambio juegan un papel crucial, a menudo se incorporan en el algoritmo de aprendizaje, ya sea para monitorizar la consistencia de partes del modelo de aprendizaje o la consistencia del modelo completo. En estos detectores se imponen frecuentemente restricciones similares a las de los algoritmos de aprendizaje incrementales, estas restricciones se relacionan principalmente con la complejidad temporal y espacial de las mismas. En general, dados los límites de la complejidad temporal y espacial, así como la dificultad de estimar los cambios en las distribuciones arbitrariamente, estos detectores de cambio a menudo operan sobre flujos de valores reales que corresponde a una medida de rendimiento del modelo de aprendizaje a lo largo del tiempo. Entonces, el problema de la detección de cambio de concepto se reduce a estimar los cambios en la distribución de un flujo de valores reales.

Los detectores de cambios emiten tres señales diferentes de cambio durante el proceso de aprendizaje. Estas señales son *estable* cuando el concepto actual permanece estable, *alerta* cuando es probable que se aproxime un cambio, y *cambio* cuando se detecta el cambio. En este trabajo se aprovechan eficientemente estas tres señales en los algoritmos propuestos. A continuación se discuten las principales características de los detectores utilizados en este trabajo.

Detector de cambios DDM (*Drift Detection Method*)

DDM (Gama et al., 2004) monitoriza el número de errores producidos por el modelo de aprendizaje y considera que la tasa de error sigue una distribución binomial. En una muestra de N casos, la tasa de error es la probabilidad de ejemplos mal clasificados μ_i con desviación estándar dada por $\sigma_i = \sqrt{\mu_i (1-\mu_i)/i}$ para cada punto i de la secuencia de ejemplos. De acuerdo con el modelo de aprendizaje PAC (Mitchell, 1997), si la distribución de instancias es estacionaria, la tasa de error disminuye a medida que aumenta el número de instancias, por lo que un aumento significativo en la tasa de error durante el entrenamiento implica un cambio en la distribución. Los valores de μ_i y σ_i son almacenados cuando $\mu_i + \sigma_i$ alcanza su valor mínimo durante el proceso, obteniéndose $\mu_{\min}$ y $\sigma_{\min}$. Con estos valores almacenados se definen tres niveles de la manera siguiente:

- *estable*, $\mu_i + \sigma_i < \mu_{\min} + \alpha \cdot \sigma_{\min}$; en este nivel la distribución es estable y se afirma que no hay cambio.
- *alerta*, $\mu_i + \sigma_i \geq \mu_{\min} + \alpha \cdot \sigma_{\min}$; en este nivel las instancias se almacenan para un posible cambio en la distribución. Sin embargo, si el cambio no se confirma después del nivel de alerta, DDM considera que hay una falsa alarma y elimina las instancias almacenadas durante esta etapa.
- *cambio*, $\mu_i + \sigma_i \geq \mu_{\min} + \beta \cdot \sigma_{\min}$, en este nivel el cambio está confirmado. Por lo que el modelo de aprendizaje es reiniciado.

Detector de cambios EDDM (*Early Drift Detection Method*)

La idea principal de este algoritmo es considerar la distancia entre dos errores consecutivos de clasificación. Este enfoque supone que si la distribución de las instancias es estacionaria, el modelo de aprendizaje mejorará su predicción y la distancia del error aumentará a medida que aumenta el número de instancias. Por lo tanto, una disminución significativa en la distancia del error implica un cambio de concepto. EDDM calcula la distancia media entre dos errores μ_i y su desviación estándar σ_i y los compara con $\mu_{\min}$ y $\sigma_{\min}$ alcanzados cuando la distribución de las distancias entre los errores es máxima. EDDM define tres niveles de la manera siguiente:

- *estable*, $(\mu_i + 2 \cdot \sigma_i) / (\mu_{\min} + 2 \cdot \sigma_{\min}) \geq \alpha$; en este nivel la distribución es estable y se afirma que no hay cambio.
- *alerta*, $(\mu_i + 2 \cdot \sigma_i) / (\mu_{\min} + 2 \cdot \sigma_{\min}) < \alpha$; en este nivel las instancias se almacenan para un posible cambio en la distribución. Sin embargo, si el cambio no se confirma después del nivel de alerta, EDDM considera que hay una falsa alarma y elimina las instancias almacenadas durante esta etapa.
- *cambio*, $(\mu_i + 2 \cdot \sigma_i) / (\mu_{\min} + 2 \cdot \sigma_{\min}) < \beta$, en este nivel el cambio está confirmado. Por lo que el modelo de aprendizaje es reiniciado. Los valores de α y β son 0,95 y 0,9 respectivamente.

Detector de cambios ADWIN (*Adaptive Sliding Window*)

ADWIN (Bifet y Gavalda, 2007) es otro detector de cambio que utiliza una ventana de detección. El enfoque mantiene una ventana deslizante W con las instancias más recientes. La idea principal de ADWIN es que si dos sub-ventanas suficientemente grandes de W tienen promedios

suficientemente distintos, se puede concluir que los valores esperados correspondientes son diferentes, y la parte más antigua de la ventana es eliminada. Esto implica responder una hipótesis estadística: ¿Ha permanecido constante la media $\mu_{\hat{W}}$ en W con nivel de confianza δ?

La parte clave del algoritmo radica en la definición del corte ε_{cut} y la prueba utilizada. Sea N el tamaño de W, n_0 y n_1 los tamaños de las sub-ventanas W_0 y W_1, de manera que $n = n_0 + n_1$. Sean $\mu_{\hat{W}_0}$ y $\mu_{\hat{W}_1}$ las medias de los valores en W_0 y W_1. El valor de corte se propone de la manera siguiente:

$$\varepsilon_{\text{cut}} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4n}{\delta}} \tag{1.1}$$

donde $m = \frac{2}{1/n_0 + 1/n_1}$ y $\delta \in (0,1)$ es el nivel de confianza establecido por el usuario, el valor recomendado es 0,2.

De acuerdo con los autores Bifet y Gavalda (2007), se detecta un cambio cuando:

$$\left|\mu_{\hat{W}_0} - \mu_{\hat{W}_1}\right| \geq \varepsilon_{\text{cut}} \tag{1.2}$$

entonces las partes más antiguas de W se reducen progresivamente hasta obtener:

$$\left|\mu_{\hat{W}_0} - \mu_{\hat{W}_1}\right| < \varepsilon_{\text{cut}} \tag{1.3}$$

En ADWIN, se utiliza una prueba estadística para comprobar un punto de corte en el que W puede dividirse en dos sub-ventanas adyacentes W_0 y W_1 que presenten suficiente diferencia, es decir, sus medias difieran en más del umbral de corte ε_{cut}. Cuando se desarrolla la condición de cambio en la ecuación 1.2, se obtiene:

$\mu_{\hat{W}_0} - \mu_{\hat{W}_1} \geq \varepsilon_{\text{cut}}$ o $\mu_{\hat{W}_0} - \mu_{\hat{W}_1} \leq -\varepsilon_{\text{cut}}$

se tiene: $\varepsilon_{cut} \geq 0$, por lo que se detecta un cambio si:

$$\mu_{\hat{W}_0} - \mu_{\hat{W}_1} \leq -\varepsilon_{\text{cut}} \leq 0 \tag{1.4}$$

o si:

$$\mu_{\hat{W}_0} - \mu_{\hat{W}_1} \geq \varepsilon_{\text{cut}} \geq 0 \tag{1.5}$$

Por lo tanto, se detecta un cambio de concepto si una de las dos condiciones definidas en las ecuaciones 1.4 y 1.5 se satisface:

- En la primera condición se tiene: $\mu_{\hat{W}_0} \leq \mu_{\hat{W}_1}$ es decir, la media del error ha aumentado en la

ventana reciente W_1 lo que implica que el número de errores está aumentando con el tiempo. En este caso, cuando ADWIN detecta un cambio, está relacionado con la disminución del rendimiento y puede considerarse como un cambio de concepto.

- En la segunda condición se tiene: $\hat{\mu}_{W_0} \geq \hat{\mu}_{W_1}$ es decir, la media del error ha disminuido en la ventana reciente W_1 lo que implica que el modelo de aprendizaje está mejorando su rendimiento. En este caso, ADWIN considera erróneamente este cambio como un cambio de concepto.

ADWIN tiene garantías probabilísticas de rendimiento en términos de la tasa de falsos positivos y falsos negativos, pero no procesa los datos de entrada con complejidad temporal y espacial constante.

Detector de cambio HDDM (*Hoeffding-based Drift Detection Method*)

$\text{HDDM}_{A-\text{test}}$ (Frias-Blanco et al., 2015) procesa cada valor entrante con una complejidad computacional constante y proporciona garantías matemáticas para las tasas de falsos positivos y falsos negativos. $\text{HDDM}_{A-\text{test}}$ ajusta automáticamente el tamaño de la ventana para tratar las distribuciones no estacionarias. Expande la ventana que contiene los valores entrantes cuando se estima una distribución estable y la reduce cuando los datos cambian. Un corolario propuesto por Hoeffding (1963) se utiliza para detectar cambios en el valor medio. El valor medio se calcula a partir de un flujo de valores que se corresponden con variables aleatorias independientes.

Dada una secuencia de variables aleatorias independientes $X_1, X_2, \ldots, X_{\text{cut}}, X_{\text{cut}+1}, \ldots, X_n$, acotadas en el intervalo $[a,b]$ y el problema de detectar un cambio significativo en la media de esta secuencia (por ejemplo, un incremento de la tasa de error de un algoritmo de aprendizaje). $\text{HDDM}_{A-\text{test}}$ estima un punto de corte relevante (cut), para después hacer una prueba estadística sobre las muestras $X_1, X_2, \ldots, X_{\text{cut}}$ y $X_{\text{cut}+1}, \ldots, X_n$. Un punto de corte relevante en esta secuencia es mín $\left\{\overline{X}_i + \varepsilon_i\right\}$ $(1 \leq i \leq n)$ (Baena-Garcia et al., 2006; Gama et al., 2004; Montgomery, 2007), donde $\overline{X}_i = \frac{1}{i}\sum_{j=1}^{i} X_j$ es la media, $\varepsilon_i = \sqrt{\frac{1}{2i}\ln\frac{1}{\alpha}}$ es su intervalo de confianza correspondiente calculado a partir de la desigualdad de Hoeffding y $1-\alpha$ es la confianza para la estimación del intervalo. Para $\overline{X} = \frac{1}{\text{cut}}\sum_{i=1}^{\text{cut}} X_i$ y $\overline{Y} = \frac{1}{n-\text{cut}}\sum_{i=\text{cut}+1}^{n} X_i$, $\text{HDDM}_{A-\text{test}}$ considera la hipótesis nula $H_0 : E\left[\overline{X}\right] \geq E\left[\overline{Y}\right]$ contra la alternativa $H_1 : E\left[\overline{X}\right] < E\left[\overline{Y}\right]$, siendo $\overline{Y} - \overline{X} \geq \varepsilon_\alpha$ la regla para rechazar H_0, donde $(n_1 = \text{cut}, n_2 = n - \text{cut})$

$$\varepsilon_\alpha = (b-a)\sqrt{\frac{n_1^{-1} + n_2^{-1}}{2}\ln\frac{1}{\alpha}} \tag{1.6}$$

Frias-Blanco et al. (2015) le llamaron a esta prueba A-test y acotaron las probabilidades de error de tipo I y tipo II para esta prueba estadística como sigue.

Corolario 1 (A-test (Frias-Blanco et al., 2015))**.** *Si* $X_1, X_2, \ldots, X_{n_1}, Y_1, Y_2, \ldots, Y_{n_2}$ *son variables aleatorias independientes con valores en el intervalo* $[a,b]$*, y si* $\overline{X} = \frac{1}{n_1}\sum_{i=1}^{n_1} X_i$*,* $\overline{Y} = \frac{1}{n_2}\sum_{i=1}^{n_2} Y_i$*, entonces para* ε_α *definido en la ecuación (1.6):*

1. si $E\left[\overline{X}\right] \geq E\left[\overline{Y}\right]$, entonces $\Pr\left\{\overline{Y} - \overline{X} \geq \varepsilon_\alpha\right\} \leq \alpha$ (cota para el error de tipo I),

   a) si $E\left[\overline{X}\right] \leq E\left[\overline{Y}\right] - \varsigma$, y $\varsigma > \varepsilon_\alpha$ entonces

   $$\Pr\left\{\overline{Y} - \overline{X} < \varepsilon_\alpha\right\} \leq e^{\frac{-2(\varsigma-\varepsilon_\alpha)^2}{\left(n_1^{-1}+n_2^{-1}\right)(b-a)^2}}$$

   (cota para el error de tipo II).

De aquí se puede derivar una prueba de dos colas por medio de la hipótesis nula $H_0 : E\left[\overline{X}\right] = E\left[\overline{Y}\right]$ contra la alternativa $H_1 : E\left[\overline{X}\right] \neq E\left[\overline{Y}\right]$, siendo $\overline{Y} - \overline{X} \geq \varepsilon_\alpha$ la regla para rechazar H_0, donde

$$\varepsilon'_\alpha = (b-a)\sqrt{\frac{n^{-1}+m^{-1}}{2}\ln\frac{2}{\alpha}}$$

En este sentido, dada una secuencia de variables aleatorias $X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m$, se pueden detectar cambios en la media poblacional monitorizando la diferencia entre promedios a partir de la prueba A-test. HDDM$_{A-\text{test}}$ define dos valores diferentes de confianza, uno correspondiente al nivel de alerta (α_W) y otro correspondiente al nivel de cambio (α_D).

1.4. Algoritmos de aprendizaje incremental

En esta sección se explican brevemente varios algoritmos que han sido utilizados para la clasificación de flujos de datos. Algunos de los algoritmos más populares propuestos para la minería de datos tradicional cumplen con los requisitos de minería de flujo. En los siguientes párrafos presentamos algunos algoritmos de aprendizaje tales como: redes neuronales, Naive Bayes, métodos de vecinos más cercanos y árboles de decisión. Varios de estos modelos se han utilizado como clasificadores base en los algoritmos que combinan clasificadores.

Redes neuronales artificiales

Las redes neuronales artificiales han sido exitosamente utilizadas en la minería de datos. En las aplicaciones tradicionales de minería de datos las redes neuronales son entrenadas utilizando un conjunto de entrenamiento fijo. Sin embargo, la mayoría de los modelos de redes neuronales requieren varias pasadas sobre el conjunto de entrenamiento para ajustar los pesos de las neuronas, usualmente se utiliza el algoritmo *backpropagation*. Sin embargo en la minería de flujos de datos los ejemplos son vistos una sola vez por el modelo y es necesario mantener acotado el costo computacional para actualizar los pesos de las neuronas con una sola pasada sobre los datos. Un ejemplo para adaptar una red para aprender a partir de flujos datos es utilizar un mecanismo de olvido, es decir cuando la red no se entrena con los mismos datos múltiples veces, esta se ajustará dinámicamente a los datos que se van obteniendo en el tiempo, por tanto pueden reaccionar de forma natural al cambio de concepto. Uno de los sistemas más conocidos capaz de manipular cambio es FRANN (*Floating Rough Approximation in Neural Network*) (Kubat y Widmer, 1995). FRANN realiza un mapeo no lineal del espacio de las instancias $\mathscr{R}^n$ a $\mathscr{R}^m$ de m neuronas ocultas de funciones de base radial. Usando una filosofía de ventana, el sistema toma cada uno de los p ejemplos en la ventana y trata de crear una neurona oculta quitando al ejemplo seleccionado. Se selecciona la neurona que provea mayor precisión sobre la ventana de ejemplos.

Naive Bayes

Este modelo se basa en el teorema de Bayes y calcula la probabilidad condicional de cada una de las clases para cada ejemplo nuevo de entrenamiento asumiendo que los atributos son independientes dada la clase. El algoritmo Naive Bayes aprende eficientemente a partir de flujos de datos no estacionarios con complejidad computacional constante.

De acuerdo con lo planteado en la sección 1.3.1 cada ejemplo de entrenamiento $e_i = (\overrightarrow{x}, y_i)$ está formado por un vector $\overrightarrow{x}$ y un valor discreto y_i, llamado etiqueta y tomado de un conjunto $\mathscr{Y}$ denominado clase. El clasificador Naive Bayes se basa en encontrar la clase más probable que describa a ese ejemplo. La clase más probable será la que cumpla:

$$y_p = argmax_{y_i \in \mathscr{Y}} P(y_i \mid \overrightarrow{x}) \tag{1.7}$$

es decir dada la probabilidad de los atributos que describen el ejemplo e_i, éste pertenezca a

la clase y_i. Por el teorema de Bayes:

$$y_p = argmax_{y_i \in \mathscr{Y}} \frac{P(\overrightarrow{x} \mid y_i) P(y_i)}{P(\overrightarrow{x})} \tag{1.8}$$

$$y_p = argmax_{y_i \in \mathscr{Y}} P(\overrightarrow{x} \mid y_i) P(y_i) \tag{1.9}$$

Se puede estimar $P(y_i)$ contando las veces que aparece y_i en el conjunto de entrenamiento y dividiéndolo por el número total de ejemplos que forman este conjunto. Para estimar el término $P(\overrightarrow{x} \mid y_i)$ se debe recorrer todo el conjunto de entrenamiento, lo que requiere un alto costo computacional. Naive Bayes simplifica el problema asumiendo que los atributos son independientes dada la clase. Así la probabilidad de que un ejemplo no etiquetado $\overrightarrow{x}$ pertenezca a la clase y_i está dada por:

$$P(\overrightarrow{x} \mid y_i) = \prod_j P\left(x_j \mid y_i\right) \tag{1.10}$$

La estimación de los parámetros del modelo (por ejemplo, $P(y_i)$ y $P(\overrightarrow{x})$) se pueden aproximar con frecuencias relativas del conjunto de entrenamiento. Estas son las estimaciones de máxima verosimilitud de las probabilidades. Las probabilidades de las clases se pueden calcular asumiendo clases equiprobables. Para estimar las probabilidades de cada uno de los atributos generalmente se utilizan la distribución de binomial para atributos discretos y la distribución normal cuando se trata con atributos continuos.

Vecinos más cercanos

Los clasificadores de vecinos más cercanos, proporcionan una manera precisa de aprender datos de forma incremental. Cada ejemplo procesado se almacena y sirve como referencia para nuevos puntos de datos. La clasificación se basa en las etiquetas de los ejemplos históricos más cercanos. A la versión sin pérdidas del algoritmo de vecino más cercano se le llama IB1, en el cual el conjunto de referencia crece con cada ejemplo aumentando los requisitos de memoria y el tiempo de clasificación. Un método más reciente de esta familia llamado IB3, limita el número de puntos de datos históricos almacenados sólo a los más «útiles» para el proceso de clasificación. Aparte de reducir los requisitos de tiempo y memoria, la limitación de tamaño del conjunto de referencia proporciona un mecanismo de olvido ya que elimina ejemplos obsoletos del modelo. Otra propuesta es el algoritmos SAM + kNN (del inglés, *Self Adjusting Memory*), que recibió el premio al mejor artículo en la Conferencia Internacional IEEE 2016 sobre Minería

de Datos (Losing et al., 2016). SAM + kNN puede manejar diferentes tipos y tasas de cambio, utilizando modelos de memoria de inspiración biológica. La idea básica de SAM + kNN es construir modelos para los conceptos actuales y anteriores.

Reglas de decisión

Los modelos basados en reglas también se pueden ajustar a entornos de flujo de datos. Los clasificadores de reglas de decisión consisten en reglas componentes disjuntas del modelo que pueden ser evaluadas aisladamente y eliminadas del modelo sin una interrupción importante. Sin embargo, las reglas pueden ser costosas desde el punto de vista computacional, ya que muchas reglas pueden afectar la decisión de un solo ejemplo. Estas observaciones sirvieron de base para el desarrollo de complejos sistemas de minería de flujo de datos como SCALLOP (Ferrer-Troyano et al., 2004), FACIL (Ferrer et al., 2005) y FLORA (Widmer y Kubat, 1996). Estos sistemas aprenden las reglas de forma incremental y emplean ventanas dinámicas para proporcionar mecanismos de olvido. Otro de los algoritmos conocidos basado en reglas es STAGGER (Schlimmer y Granger, 1986), el cual induce fórmulas booleanas que se mueven desde caracterizaciones más generales (disyunción de pares atributo-valor) a más particulares (conjunciones de pares atributo-valor). Sobre estas caracterizaciones son definidos tres operadores de búsqueda (para crear una fórmula más general, una más particular, o negar una fórmula existente), los cuales son ejecutados cuando el modelo comete un error en la clasificación.

Árboles de decisión

La construcción de árboles de decisión ha sido ampliamente estudiada en la comunidad del aprendizaje automático. En la minería de flujos de datos, la desigualdad de probabilidades de Hoeffding (Del Campo Ávila, 2007; Frias-Blanco et al., 2015; Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno, 2016), de Chernoff (Del Campo Ávila, 2007; Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno, 2016) han sido herramientas poderosas para la inducción incremental de árboles de decisión. La cota Hoeffding indica que con probabilidad $1-\delta$, la media real de una variable aleatoria con rango R no difiere de la media estimada después de n observaciones independientes por más de:

$$\varepsilon = \sqrt{\frac{R^2 ln\left(\frac{1}{\delta}\right)}{2n}} \tag{1.11}$$

Utilizando esta cota, Domingos y Hulten (2000) propusieron VFDT (*Very Fast Decision*

Tree). VFDT induce incrementalmente un árbol de decisión a partir de un flujo de datos, sin la necesidad de almacenar ejemplos de entrenamiento después de haber sido utilizados para actualizar el árbol. Este algoritmo es similar a los algoritmos clásicos de inducción de árboles de decisión (Breiman et al., 1984; Quinlan, 1986, 2014), solo difiere en la selección del atributo para dividir. En lugar de seleccionar el mejor atributo después de ver todos los ejemplos, VFDT utiliza la cota de Hoeffding para determinar cuántas instancias deben ser muestreadas antes de poder elegir, con probabilidad $1-\delta$, qué atributo debe ser usado para la expansión de cada nodo. El objetivo es asegurar que, con una alta probabilidad, el atributo elegido usando n ejemplos (donde n es tan pequeño como sea posible) es el mismo que sería elegido usando ejemplos infinitos. Sea $G(X_i)$ la medida heurística utilizada para elegir atributos de prueba (por ejemplo, la medida podría ser ganancia de información como en C4.5). Supongamos que G debe ser maximizada, y sea X_a el atributo con el G más alto observado después de ver n ejemplos y X_b el segundo mejor atributo. Sea $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b) \geq 0$ la diferencia entre sus valores heurísticos observados. Entonces, dado un δ deseado, la cota de Hoeffding garantiza que X_a es la elección correcta con probabilidad $1-\delta$ si se han visto n ejemplos en este nodo y $\Delta\bar{G} > \varepsilon$. En otras palabras si $\Delta\bar{G} > \varepsilon$ entonces la cota de Hoeffding garantiza $\Delta G > \Delta\bar{G} - \varepsilon > 0$ con probabilidad $1-\delta$, y por lo tanto que X_a es de hecho el mejor atributo con probabilidad $1-\delta$. Esto es válido siempre y cuando el valor $\bar{G}$ de un nodo pueda verse como un promedio de valores G para los ejemplos en ese nodo, como es el caso de las medidas típicamente usadas. Así, un nodo necesita acumular ejemplos del flujo de datos hasta que ε sea menor que $\Delta\bar{G}$.

Se han propuesto muchas mejoras al algoritmo VFDT básico. Domingos y Hulten (2000) proponen un método para limitar el uso de la memoria. Propusieron eliminar las estadísticas de las hojas «menos prometedoras». Los nodos menos prometedores se definen como los que tienen los valores más bajos de $p_l e_l$, donde p_l es la probabilidad de que los ejemplos lleguen a una hoja en particular l, y e_l es la tasa de error observada en l. Para reducir aún más el uso de memoria, también sugirieron la eliminación de las estadísticas de los atributos más pobres de cada hoja.

La cota Hoeffding es válida para cualquier tipo de distribución. Una desventaja de ser tan general es que es más conservadora que una cota dependiente de la distribución y, por lo tanto, requiere más ejemplos de lo que realmente es necesario.

Un enfoque diferente para manipular atributos numéricos fue propuesto por Gama et al. (2003). Este enfoque utiliza árboles binarios como una forma de discretizar dinámicamente los valores numéricos. El mismo trabajo también investiga el uso de un clasificador adicional en los nodos hoja, en este caso Naive Bayes. En este trabajo también se investiga la posibilidad de agregar otros algoritmos como clasificadores en los nodos hojas con el objetivo de mejorar el

rendimiento de VFDT.

El algoritmo VFDT original fue diseñado para flujos de datos estáticos y no proporcionó ningún mecanismo de olvido. El problema de clasificar los flujos de datos no estacionarios con árboles Hoeffding fue abordado primero por Hulten et al. (2001). Los autores propusieron un nuevo algoritmo llamado CVFDT, que utilizó una ventana de tamaño fijo para determinar qué nodos están envejeciendo y pueden necesitar una actualización. Para las partes del árbol Hoeffding que se hacen viejos e inexactos, se crean sub-árboles alternativos que posteriormente reemplazan a los nodos obsoletos. Vale la pena señalar, que todo el proceso no requiere un re-entrenamiento del modelo. Los ejemplos obsoletos se olvidan al actualizar las estadísticas de los nodos y los cambios de modelo necesarios se realizan en los sub-árboles en lugar del clasificador completo.

Diferentes enfoques para agregar un mecanismo de olvido al algoritmo VFDT incluyen el uso de los detectores de cambios EWMA (de inglés, *Exponential Weight Moving Average*) y ADWIN (Bifet, 2009). Este último, ofrece garantías de rendimiento en relación con la tasa de error obtenida y ambos métodos mencionados son más precisos y consumen menos memoria que CVFDT. El costo de las extensiones de árbol EWMA y ADWIN es el tiempo promedio necesario para procesar un solo ejemplo.

VFDT junto con Naive Bayes son dos de los algoritmos más utilizados como clasificadores bases en la mayoría de los algoritmos que combinan clasificadores.

1.5. Algoritmos para combinar clasificadores

Los algoritmos que combinan clasificadores han recibido especial atención en la comunidad de flujo de datos (Krawczyk et al., 2017). Estos algoritmos combinan la predicción de los clasificadores bases y se pueden clasificar por la forma en que actualizan su clasificadores bases. La primera clasificación está dada por la utilización de bloques de instancias. Los ensambles basados en bloques de instancias dividen el flujo de datos en pequeños bloques de igual tamaño y entrenan clasificadores con cada uno de estos bloques. La adaptación a los cambios de concepto de estos ensambles generalmente es lenta porque tienen que esperar a que se llene el bloque para actualizar los clasificadores bases. La segunda clasificación de los ensambles es por la actualización de los clasificadores de forma incremental. Estos métodos utilizan clasificadores incrementales como clasificadores bases, es decir actualizan los modelos en la medida que se obtienen las instancias. En las secciones se presentan las principales características de algunos de los algoritmos más conocidos en la minería de flujos de datos.

1.5.1. Multiclasificadores que utilizan bloques de instancias

En esta sección se realiza una revisión de los principales algoritmos que dividen el flujo de datos en bloques de igual tamaño y entrenan clasificadores con cada uno de estos bloques. El esquema general seguido por estos algoritmos se muestra en Algoritmo 1.

Algoritmo 1: Multiclasificadores basados en bloques de instancias.

Entrada:
flujo de datos, cantidad M de clasificadores bases, tamaño N del bloque de instancias

Salida :
clase predicha por el ensamble

```
1 Bloque : bloque de instancias para crear clasificadores
2 foreach instancia de entrenamiento do
3     if Bloque no está lleno then
4         Bloque+ = nuevainstancia
5     else if Bloque está lleno then
6         realizar acciones como, crear nuevo clasificador, actualizar clasificadores, calcular
            pesos, etc.
7         Vacciar el Bloque
8 return en todo momento la clase predicha por el multiclasificador
```

SEA (*Streaming Ensemble Algorithm*)

Uno de los primeros algoritmos de ensamble para el procesamiento de flujos de datos fue SEA (*Streaming Ensemble Algorithm*) (Street y Kim, 2001). SEA crea los clasificadores base a partir de pequeños subconjuntos de los datos, leídos secuencialmente en bloques de un tamaño fijo. El tamaño de esos bloques es un parámetro importante porque es responsable del equilibrio entre precisión y flexibilidad. Cada bloque se utiliza para entrenar un nuevo clasificador, es cual es comparado con los demás miembros del conjunto. Si alguno de los miembros es peor que el nuevo clasificador este se sustituye por el nuevo. Para evaluar los clasificadores, Street y Kim proponen utilizar la precisión de clasificación obtenida en el bloque de datos más reciente. El algoritmo cuenta con un límite máximo de clasificadores que actúa como mecanismo de adaptación. Al ser alcanzado este límite máximo obliga al algoritmo sustituir a clasificadores básicos anteriores siguiendo cierto criterio de remplazo. Para combinar las predicciones de los clasificadores base utiliza votación por mayoría no ponderada y como clasificador base utiliza el

algoritmo C4.5. SEA presenta problemas para adaptarse a los cambios de conceptos abruptos; en estos resultados influye el mecanismo de votación utilizado ya que los clasificadores dejan de aprender una vez que son creados.

AWE (*Accuracy Weighted Ensemble*)

Bajo el mismo esquema de entrenamiento de SEA, Wang et al. (2003) propusieron el método AWE (*Accuracy Weighted Ensemble*). AWE entrena un nuevo clasificador C' en cada bloque de datos entrantes y usan ese bloque para evaluar todos los clasificadores del conjunto para seleccionar los mejores clasificadores. Los autores demostraron que para un conjunto de clasificadores $\mathscr{E}_k$ construido a partir de los k bloques de datos más recientes y un único clasificador G_k también construido a partir de estos k bloques , se cumple el siguiente teorema:

Teorema. *$\mathscr{E}_k$ produce un error de clasificación menor que G_k, si los clasificadores en $\mathscr{E}_k$ son pesados de acuerdo a su precisión de clasificación esperada en los datos de la prueba.*

De acuerdo con este teorema , para pesar adecuadamente a los clasificadores base se necesita conocer la función real que se está aprendiendo, la cual no está disponible. Es por eso que los autores proponen derivar los pesos mediante la estimación de la tasa de error en los datos más recientes x_i, como se muestra en las ecuaciones siguientes:

$$\mathrm{MSE}_i = \frac{1}{|x_i|} \sum_{(x,y)\in x_i} \left(1 - f_y^i(x)\right)^2 \tag{1.12}$$

$$\mathrm{MSE}_r = \sum_y p(y)\left(1 - p(y)\right)^2 \tag{1.13}$$

$$w_i = \mathrm{MSE}_r - \mathrm{MSE}_i \tag{1.14}$$

La función $f_y^i(x)$ denota la probabilidad dada por el clasificador C_i que x sea una instancia de la clase y. Esta es una característica interesante del cálculo del peso, ya que la mayoría de las funciones de ponderación utilizan sólo la predicción de los clasificadores en lugar de la probabilidad de todas las clases posibles. También es importante notar que para el clasificador candidato denotado como C', la tasa de error se calcula utilizando validación cruzada en el bloque actual para evitar el sobre-ajuste. El valor de MSE_r es el error medio cuadrático de un clasificador y se utiliza para poner a cero los pesos de los modelos que no contienen ningún conocimiento útil sobre los datos.

Para los primeros k bloques de datos, el algoritmo crea los clasificadores disponibles, pero al procesar otros bloques, selecciona sólo los k mejores clasificadores para conformar la combinación. Los autores plantean que para un flujo de datos grande es imposible almacenar todos los clasificadores creados durante el proceso de aprendizaje y la selección no se puede realizar en un conjunto ilimitado de clasificadores. Es por eso que para los flujos de datos dinámicos, es necesario introducir un nuevo parámetro que limite el número de clasificadores disponibles para la selección.

El algoritmo AWE funciona bien en los flujos de datos con conceptos recurrentes, así como en diferentes tipos de cambio. Al igual que SEA, es crucial definir correctamente el tamaño de los bloques de datos, ya que determina la flexibilidad del multiclasificador. Aunque como en SEA, su rendimiento frente a cambios de conceptos graduales es aceptable, pero esto no ocurre así cuando los cambios son abruptos. Eso se debe fundamentalmente a que AWE espera al próximo bloque de entrenamiento para actualizar los pesos de los clasificadores base.

FAE (*Fast Adapting Ensemble*)

El algoritmo FAE propuesto por Ortíz Díaz et al. (2014) es un método de combinación de clasificadores diseñado para adaptarse rápidamente a los cambios de concepto, tanto abruptos como graduales, y se especializa en el tratamiento de conceptos recurrentes. Esta propuesta cuenta con una colección de clasificadores que representan a varios de los conceptos aprendidos previamente; como diferencia, en este caso los clasificadores están organizados en activos e inactivos según su comportamiento frente a los datos actuales. FAE es un multiclasificador que toma su decisión global a partir de las decisiones parciales de los clasificadores activos; mientras que conserva un grupo de clasificadores inactivos como almacén de antiguos conceptos, los cuales le favorecen en el tratamiento de conceptos recurrentes. Estos clasificadores inactivos son activados de forma muy rápida si reaparece el concepto al cual ellos representan. La reactivación de clasificadores y la inserción, de ser necesaria, de nuevos clasificadores actualizados garantiza una rápida adaptación, sobre todo si existen conceptos recurrentes.

Este algoritmo divide el flujo de datos de entrenamiento en bloques de igual tamaño y con cada uno de estos bloques construye, de ser necesario, un nuevo clasificador básico que agrega al multiclasificador; de esta forma extrae conocimiento de grandes conjuntos de datos de forma natural. El algoritmo FAE también fija un límite máximo de clasificadores a almacenar, que al ser alcanzado y como mecanismo de adaptación obliga a sustituir a clasificadores base anteriores siguiendo ciertos criterios de reemplazo. Esta es la única forma de eliminar clasificadores que tiene el algoritmo.

1.5.2. Multiclasificadores incrementales

Los ensambles incrementales (ver algoritmo 2) utilizan como clasificadores bases algoritmos incrementales, es decir se van actualizando en la medida en la que se obtienen los datos. Estos algoritmos están diseñados para aplicaciones con limitaciones estrictas de tiempo y memoria, o aplicaciones en las que no se puede permitir procesar cada ejemplo de entrenamiento más de una vez, por ejemplo, aplicaciones en las que la cantidad de datos entrantes es muy grande (potencialmente infinita) (Krawczyk et al., 2017). Los ensambles incrementales han recibido más atención por parte de los investigadores que sus contrapartes basados en bloques de ejemplos. Esto se debe principalmente a las ventajas del aprendizaje incremental y su aplicación en varios escenarios de la vida real. A continuación se realiza una revisión de las propuestas más representativas en esta área.

Algoritmo 2: Ensambles incrementales.

Entrada:
flujo de datos, cantidad M de clasificadores

Salida :
clase predicha por el ensamble

```
1 foreach instancia de entrenamiento do
2     aculizar los clasificadores bases con cada instancia de entrenamiento
3 return en todo momento la clase predicha por el ensamble
```

Bagging y Boosting

Bagging (Breiman, 1996) y Boosting (Freund et al., 1996) han sido dos de los algoritmos más extendidos y sobre los que se han desarrollado muchas variantes. Ambos algoritmos combinan clasificadores con el objetivo de mejorar la predicción de los clasificadores individuales.

Bagging (Breiman, 1996) crea un conjunto de M clasificadores base a partir de diferentes subconjuntos de entrenamiento de tamaño N realizando muestreo con reemplazamiento a partir del original. Si los subconjuntos generados son parecidos los modelos inducidos usando un mismo algoritmo también fuesen parecidos y afectaría la diversidad entre los clasificadores individuales. Pero esto no ocurre siempre de este modo, ya que existen algoritmos de inducción que son muy sensibles a pequeñas variaciones en el conjunto de entrenamiento, por ejemplo, algunos algoritmos de inducción de árboles de decisión. Esta característica de inestabilidad es la que da nombre a los algoritmos conocidos como «aprendices débiles» y se ve acentuada cuando

el subconjunto de entrenamiento tiene un tamaño limitado. Normalmente, las predicciones de los clasificadores base se combina utilizando el voto mayoritario, es decir se elige la clase que haya sido seleccionada por la mayoría de los clasificadores individuales.

En Bagging, cada conjunto de entrenamiento de cada modelo base contiene cada uno de los ejemplos originales K veces donde $P(K = k)$ sigue una distribución Binomial. La distribución Binomial cuando $N \to \infty$ tiende a la distribución de Poisson$(1) = \exp(-1)/k!$. Utilizando este hecho Oza y Russell (2001) propusieron la versión incremental del algoritmo Bagging. Este algoritmo en vez de aplicar muestreo con reemplazamiento, le asigna a cada ejemplo de entrenamiento un peso de acuerdo con la distribución de Poisson(1).

El algoritmo Boosting, propuesto por Freund et al. (1996), asigna pesos a las experiencias en el conjunto de entrenamiento. El objetivo de este algoritmo es inducir un modelo que minimice la suma de los pesos de los ejemplos que no se clasifiquen correctamente. En un principio, los pesos de los diferentes ejemplos tendrán el mismo valor y los errores serán igual de importantes. Pero en cada iteración, los pesos de los ejemplos que han sido clasificador incorrectamente se aumentarán para darles más importancia. De esta forma, en cada iteración se le da más importancia a los ejemplos que no han sido clasificados correctamente en pasos anteriores. Para combinar las predicciones de los clasificadores base se tiene en cuenta el error de cada clasificador. Sumando los pesos de los clasificadores que eligen la misma clase, se selecciona la que acumule mayor valor.

Oza y Russell (2001) también propusieron un versión incremental del algoritmo Boosting. Este ensamble mantiene un conjunto de clasificadores de tamaño fijo entrenado con los ejemplos más recientes. Cada ejemplo nuevo se utiliza para actualizar cada clasificador de manera secuencial. Para cada ejemplo nuevo entrante, inicialmente se le asigna el mayor peso posible $\lambda = 1$. El primer clasificador se entrena con este ejemplo k veces, con $k = \text{Poisson}(\lambda)$. Luego de la actualización del clasificador con el nuevo ejemplo, si este clasifica correctamente el ejemplo $\lambda = \lambda\left(\frac{N}{\lambda^{sc}}\right)$, de lo contrario $\lambda = \lambda\left(\frac{N}{\lambda^{sw}}\right)$, donde N es la cantidad de ejemplos vistos hasta el momento, λ^{sc} es la suma de los lambdas de los ejemplos clasificados correctamente y λ^{sw} es la suma de los lambdas de los ejemplo clasificados incorrectamente. Este procedimiento se repite para cada clasificador en el ensamble utilizando el nuevo λ. El principal problema con este enfoque es que el procedimiento de actualización de peso se debe realizar al pasar de una etapa a la siguiente. En el algoritmo Boosting incremental, el procedimiento de actualización de los pesos depende de los pesos de las instancias en la etapa anterior. Sin embargo, en un flujo de datos, el número de instancias de entrenamiento puede ser infinito y, por lo tanto, no es factible controlar la etapa actual de Boosting.

Las versiones incrementales de los algoritmos Bagging y Boosting están diseñada para flujos de datos estacionarios, es decir donde no existen cambios de concepto. Ambas propuestas han sido extendidas en la literatura para adaptarse a los cambios de concepto. LeveragingBag (Bifet, Holmes y Pfahringer, 2010), OzaBagADWIN and OzaBoostADWIN son los más representativos (Bifet et al., 2009). Aunque estos algoritmos son capaces de manipular cambios de concepto, el mecanismo de adaptación se realiza solo a nivel de ensamble. Por ejemplo, LeveragingBag, OzaBagAdwin y OzaBoostAdwin monitorizan el rendimiento predictivo de cada clasificador base utilizando el detector de cambios ADWIN. Sin embargo, cuando el detector estima un cambio de concepto, solo se reemplaza el peor clasificador de base. Esto puede conducir a una adaptación ineficaz a los cambios, ya que el peor clasificador puede no ser el afectado por el cambio actual.

1.6. Consideraciones finales del capítulo

En este capítulo se han descrito de manera general los principales algoritmos y técnicas empleadas en el aprendizaje automático y extracción de conocimiento. Se ha realizado una revisión de la literatura sobre los principales algoritmos multiclasificadores, incrementales y que se adaptan al cambio de concepto. Dado que los spammers cambian la estructura de los mensajes spam para burlar a los filtros tradicionales, es importante que los filtros anti-spam sean capaces de adaptarse a estos cambios. Algunos de los algoritmos tradicionales se han adaptado para ser capaces de detectar el cambio de concepto. Como los correos electrónicos representan un flujo de datos potencialmente infinito en el tiempo, resulta apropiada la utilización de los algoritmos incrementales en lugar de algoritmos de clasificación por lotes.

CAPÍTULO 2

Modelo para la detección de spam aplicando clasificadores incrementales

En este capítulo se presenta un nuevo modelo para la detección de spam aplicando clasificadores incrementales, mediante una descripción general del modelo y de las colecciones. Para finalizar el capítulo, se realiza una concepción teórica-metodológica del procedimiento en general, la cual se encuentra estructurada en tres etapas con sus fases correspondientes. También se muestran los resultados experimentales.

2.1. Descripción general del modelo

El modelo tiene dos etapas principales: (1) creación y entrenamiento del clasificador, y (2) clasificación y retroalimentación. La primera etapa (Figura 2.1a) se realiza solo una vez (en la fase de desarrollo) e incluye los siguientes elementos:

- *Corpus*: Una colección de correos electrónicos (como mismo se obtienen del servidor) previamente clasificados (en spam o ham) de la cual se deben extraer los rasgos necesarios para la clasificación de nuevos mensajes cuya clase es desconocida.

- *Preprocesador*: Su función es de intermediario entre los correos electrónicos en bruto y el clasificador, ya que el clasificador no entiende el formato de un correo electrónico sino que espera que la información esté preprocesada en un formato como VSM o ARFF (Attribute-Relation File Format)[1]. En esta primera etapa el preprocesador es el encargado de generar

[1] http://www.cs.waikato.ac.nz/ml/weka/arff.html

un fichero ARFF, que representa la información contenida en el corpus, con el cual el clasificador se crea y se entrena.

- *Clasificador*: Es el encargado de aprender las clases spam y ham a través del conocimiento extraído de los correos electrónicos con los que fue entrenado, para posteriormente ser capaz de predecir la clase de un mensaje cuya clase se desea conocer, y con esto poder filtrar el spam.

Por otra parte, en la segunda etapa (Figura 2.1b) el modelo ya se encuentra listo para ser usado e intervienen los siguientes elementos:

- *Servidor*: Representa la parte del servidor de correos, perteneciente al usuario para el que fue configurado el modelo, en el mismo los mensajes son enviados, recibidos y almacenados, y es el encargado de proveer los nuevos mensajes al sistema para su posterior clasificación, así como los mensajes clasificados por el usuario para el entrenamiento y adaptación del sistema.

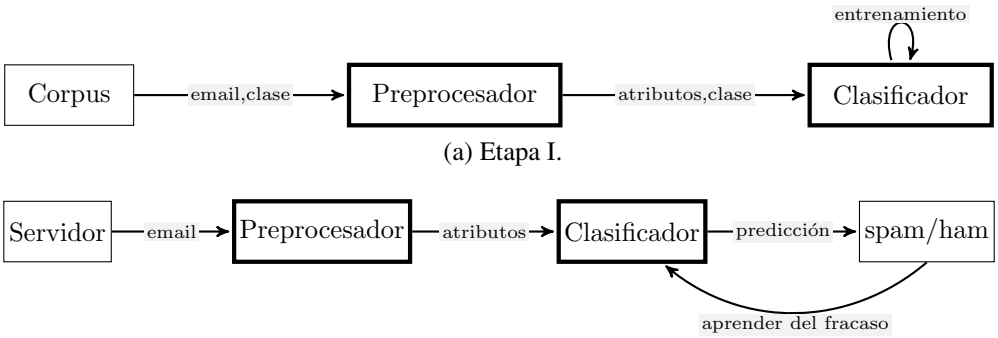
- *Preprocesador*: Su función es similar que en la primera etapa solo que esta vez en lugar de preprocesar los correos electrónicos de un corpus, los recibe del servidor de correo.

- *Clasificador*: En esta etapa el clasificador se encarga de predecir la clase de los nuevos mensajes que arriban al servidor de correo, para filtrar el spam de los correos legítimos en la bandeja de entrada del usuario. Al mismo tiempo el clasificador aprende de sus errores, es decir, cada vez que clasifica un mensaje erróneamente, se entrena en dependencia de la acción del usuario, la cual es extraída automáticamente del servidor de correos cuando es marcado como spam un mensaje que el clasificador había etiquetado como legítimo, o cuando es marcado como legítimo un mensaje que el clasificador había etiquetado erróneamente como spam.

2.2. Descripción de las colecciones

En este trabajo se utilizaron corpus públicos de SpamAssassin[2] y Enron-Spam[3]. El corpus SpamAssassin se encuentra dividido en tres partes o categorías principales *spam*, *easy_ham* y *hard_ham*:

[2] http://spamassassin.apache.org/old/publiccorpus/
[3] http://www2.aueb.gr/users/ion/data/enron-spam/index.html



(a) Etapa I.

(b) Etapa II.

Figura 2.1: Modelo general.

1. *spam*: compuesto por 500 correos electrónicos spam, todos los cuales han sido clasificados manualmente, no por filtros automáticos anti-spam.

2. *spam_2*: compuesto por 1397 correos electrónicos spam, son mensajes con las mismas características que los de la categoría *spam* solo que son mensajes más recientes.

3. *easy_ham*: compuesto por 2500 correos electrónicos ham, estos en su mayoría son mensajes que se pueden diferenciar con facilidad de mensajes spam, ya que no contienen características comunes en correos spam como HTML.

4. *easy_ham_2*: compuesto por 1400 correos electrónicos ham, son mensajes con las mismas características que los de la categoría *easy_ham* solo que son mensajes más recientes.

5. *hard_ham*: compuesto por 250 correos electrónicos ham los cuales tienen características que son usualmente encontradas en correos spam: el uso de HTML, texto coloreado, frases que podrían confundir y hacer pensar que se trata de un mensaje spam, etc.

En total este corpus cuenta con 6047 mensajes, con una proporción de spam del 31 % aproximadamente.

Por otra parte, el corpus Enron-Spam fue originalmente recolectado y preparado por el proyecto CALO (A Cognitive Assistant that Learns and Organizes). El mismo contenía correos electrónicos de cerca de 150 usuarios, de los cuales la mayoría son directivos de la compañía Enron, organizados en carpetas. Esta colección incluye una gran cantidad de mensajes personales legítimos, aproximadamente de 43000 mensajes ham y aproximadamente 50000 mensajes spam (Metsis et al., 2006). Como el objetivo de la presente investigación es la detección personalizada de spam, se utilizó el marco de trabajo preparado por Metsis et al. (2006) los cuales crearon seis

colecciones de ham, seleccionando los mensajes legítimos de los seis empleados con mayor cantidad de mensajes recibidos, y tres colecciones compuestas por spam del corpus SpamAssassin y otras fuentes.

2.3. Concepción teórica-metodológica del procedimiento general

Como parte del modelo conceptual se desarrolla un procedimiento metodológico general que incluye varios procedimientos específicos, estructurados en tres etapas con sus fases correspondientes que en su conjunto resumen el contenido del modelo. Las etapas del procedimiento general son:

1. Representar el corpus textual.
2. Clasificar los correos utilizando algoritmos incrementales.
3. Validar los resultados de la clasificación.

2.3.1. Representar el corpus textual

La detección de spam se puede realizar de manera personalizada (a nivel de usuarios individuales) (Metsis et al., 2006; Sanghani y Kotecha, 2016), o a nivel de servidor, es decir, un filtro anti-spam común a todos los usuarios. La detección personalizada a demostrado obtener buenos resultados en varios escenarios (Metsis et al., 2006; Sanghani y Kotecha, 2016), y esto se debe principalmente a que los filtros solo se entrenan con los correos de cada usuario por separado, y puede ocurrir en algunos casos que lo que es spam para un usuario no lo sea para otro. Debido a esto en esta investigación se utiliza la detección personalizada y para esto el corpus de Enron se divide en seis colecciones, una por cada empleado de la empresa. Los correos spam se seleccionaron de diferentes fuentes y se agruparon en tres colecciones. La primera colección (S1) contiene mensajes enviados entre Mayo del 2001 y julio del 2005, la segunda (S2) entre Agosto del 2004 y Julio del 2005 y la tercera (S3) entre diciembre del 2003 y Septiembre del 2005.

A partir de las seis colecciones de correos no spam (ham) de la empresa Enron se crearon los nuevos corpus aleatoriamente con cada una de las 3 colecciones de spam (S1, S2 y S3). En tres de las colecciones resultantes se utilizó una tasa de ham:spam de aproximadamente 3:1 y

Tabla 2.1: Características de las colecciones creadas.

Colecciones	DS1	DS2	DS3	DS4	DS5	DS6
ham	1966	3669	4363	4012	2364	2714
spam	3675	1499	1496	1500	4500	4496

en las otras tres se invirtió la tasa es decir 1:3. Esto implica que existe un nivel de desbalance, por lo que esto se tiene en cuenta en la Sección 2.3.3 al de evaluar los algoritmos incrementales. Por otro lado, para tener en cuenta el problema del cambio de concepto todos los mensajes fueron ordenados cronológicamente. La Tabla 2.1 muestra las características de los seis corpus obtenidos representados por DS1, DS2, ..., DS6.

Para el preprocesamiento de los cuerpos de los mensajes se aplicaron técnicas de minería de textos. Específicamente se utilizó el modelo espacio vectorial (Salton et al., 1975) para representar los documentos debido a que es ampliamente reconocido en la comunidad de minería de textos. Primero, se les aplicó a los documentos técnicas como lematización, *stemming*, eliminación de palabras vacías o *stop-words*, etc., para transformarlos en una secuencia de palabras o *tokens*, de los cuales se crea una secuencia de índices de posibles términos para cada documento durante el paso de extracción de términos. Los términos extraídos se utilizan como el vocabulario V del corpus. Un problema aquí, es que el número de términos extraídos puede ser extremadamente grande, por tanto, es necesario realizar una reducción del vocabulario. Para cada documento d_j sea crea un vector de frecuencia de términos $d_{tf} = (tf_d(t_1), \ldots, tf_d(t_m))^T$, donde $tf_d(t)$ denota el número de veces que el término $t \in V$ aparece en el documento d. Después de crear este vector para cada documento, se reduce el número de términos mediante un algoritmo de selección de rasgos. En este caso se utilizó la ganancia de la información, ya que es uno de los métodos utilizados exitosamente en la minería de textos (Liu, 2004; Yang y Pedersen, 1997).

2.3.2. Clasificación de los correos utilizando algoritmos incrementales

Los ensambles de clasificadores incrementales han sido usados satisfactoriamente para minar flujos de datos (Brzezinski y Stefanowski, 2014; Frías-Blanco, Verdecia-Cabrera, Ortiz-Díaz y Carvalho, 2016). Los métodos de ensamble combinan las predicciones de sus clasificadores base con el fin de mejorar la precisión predictiva obtenida por un único clasificador. En esta sección se analizarán tres métodos de ensambles para el filtrado de spam.

FASE (*Fast Adaptive Stacking of Ensembles*) (Frías-Blanco, Verdecia-Cabrera, Ortiz-Díaz y Carvalho, 2016) está diseñado para tratar con cambios de concepto. Para entrenar los clasi-

ficadores base, FASE usa el método de *bagging* incremental previamente mencionado (Oza y Russell, 2001) para funcionar en entornos no estacionarios. FASE emplea clasificadores adaptativos tanto en el proceso de votación como en los clasificadores base (ver Figura 2.2a). Cada clasificador adaptativo usa el método para la detección de cambios basado en Hoeffding (HDDM) (Frias-Blanco et al., 2015) para detectar cambios y estimar errores, el cual supervisa la tasa de errores con el fin de accionar tres señales de cambio diferentes durante el proceso de aprendizaje.

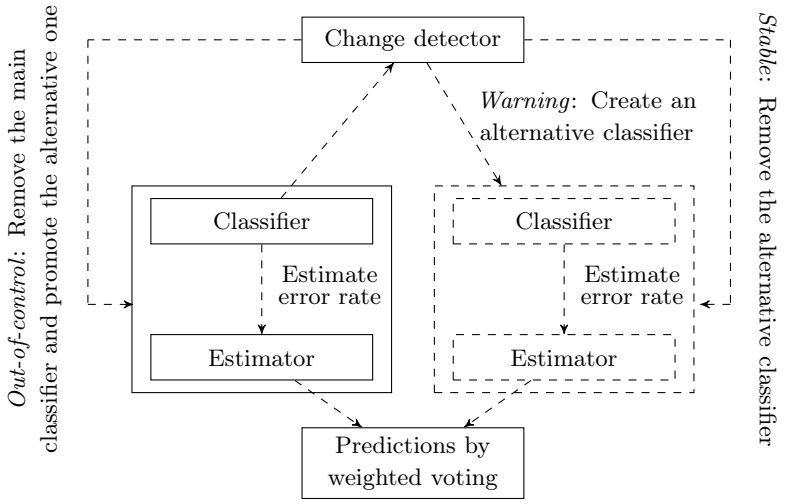
El *meta-learner* de FASE recibe meta-instancias como entrada, donde cada atributo es nominal. FASE utiliza un enfoque *test-then-train* (Gama et al., 2009) para generar meta-instancias (ver Figura 2.2b). Por lo tanto, para cada instancia de entrenamiento original I=($\overrightarrow{a}$,c), FASE genera una meta-instancia de entrenamiento $M = (\hat{c}_1, \hat{c}_2 ..., \hat{c}_n, c)$, donde cada $\hat{c}_i$ es un valor de atributo y c es su correspondiente etiqueta de clase. Cada valor de atributo $\hat{c}_i$ de la meta-instancia M corresponde a la predicción del clasificador base i para la instancia I. Para esta meta-instancia M, el valor $\hat{c}_i$ es la etiqueta de clase predicha por el clasificador i. La etiqueta de clase de la meta-instancia M es la misma que la de la instancia de entrenamiento original. Los resultados experimentales muestran que FASE puede ser una alternativa eficiente para aprender de flujos de datos.

Los algoritmos de *bagging* y *boosting* incrementales también han sido extendidos para ser capaces de tratar con flujos de datos no estacionarios; por ejemplo, LeveragingBag (Bifet, Holmes y Pfahringer, 2010) y OzaBagADWIN (Bifet et al., 2009). Básicamente, utilizan el detector de cambios ADWIN (Bifet y Gavalda, 2007) para reemplazar, cuando se detecta un cambio de concepto, el clasificador base con el rendimiento predictivo más bajo en el ensamble por un nuevo clasificador base. Por otra parte LeveragingBag, a diferencia de OzaBagADWIN, aprovecha el rendimiento de bagging, con dos mejoras de aleatoriedad: aumentar el remuestreo y usar códigos de detección de salida.

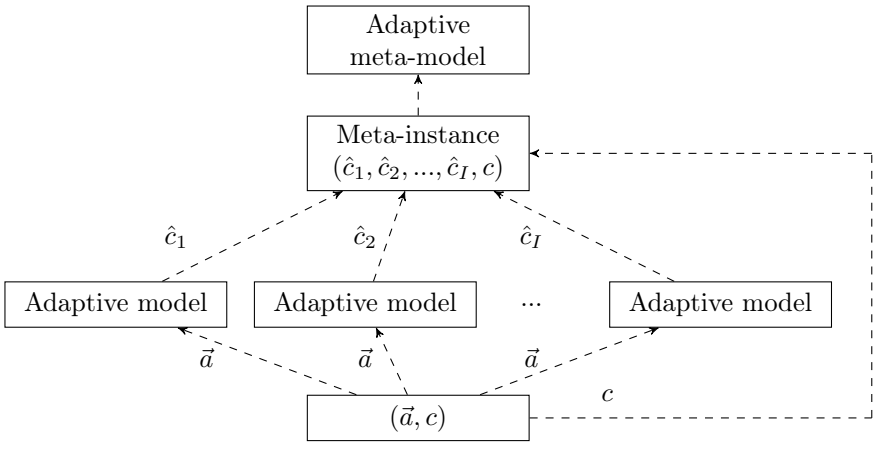
2.3.3. Resultados experimentales

El estudio experimental descrito en esta sección evalúa los algoritmos en los corpus de prueba seleccionados. Todos los experimentos fueron realizados usando MOA (Bifet, Holmes, Kirkby y Pfahringer, 2010), un marco de trabajo para el análisis incremental.

Los experimentos siguieron el enfoque *test-then-train*, el cual es derivado del error secuencial predictivo. En este enfoque, cuando una nueva instancia arriba, primeramente un modelo de clasificación predice su etiqueta de clase (paso de prueba). Seguidamente, la instancia es usada



(a) Mecanismo de aprendizaje usado en *learners* adaptativos.



(b) Esquema de FASE.

Figura 2.2: FASE.

por un algoritmo de aprendizaje para actualizar el modelo de clasificación (paso de entrenamiento) (Bifet, Holmes, Kirkby y Pfahringer, 2010).

Con cada nueva instancia, el clasificador fue probado y luego actualizado. Durante el proceso de aprendizaje, el desempeño predictivo fue evaluado usando una ventana deslizante de tamaño 100 (Bifet, Holmes, Kirkby y Pfahringer, 2010). El desempeño predictivo instantáneo de los clasificadores, calculado para cada 100 instancias, fue procesado como la fracción entre el número de instancias clasificadas correctamente en la ventana deslizante actual y el tamaño de la ventana. Teniendo en cuenta que en los corpus de prueba seleccionados existe un cierto grado de desbalance, y la medida «precisión» no provee información adecuada sobre el rendimiento de un clasificador con datos desbalanceados (He y A. Garcia, 2009), también se incluyeron las medidas «precisión» y «*recall*». La precisión es una medida de exactitud, y la recuperación es una medida de completitud. En este trabajo, se usó la clase minoritaria como la clase positiva y la clase mayoritaria como la clase negativa. La matriz de confusión fue actualizada incrementalmente usando el enfoque *test-then-train*.

Con el fin de evaluar el desempeño predictivo de los algoritmos en presencia del cambio de concepto se utilizó la siguiente definición (Bifet, Holmes, Kirkby y Pfahringer, 2010):

Definición 2. Dado dos flujos de datos *a y b,* se define $c = a \oplus_{t_0}^{W} b$ como el flujo de datos resultante de la unión de los dos flujos de datos *a* y *b,* donde t_0 es el punto de cambio*,* W es la longitud del cambio, $Pr(c(t) = b(t)) = 1/\left(1 + e^{-4(t-t_0)/W}\right)$ *y* $Pr(c(t) = a(t)) = 1 - Pr(c(t) = b(t))$.

Como consecuencia de esta definición, los nuevos conjuntos de datos fueron $DS1 \oplus_{t_0}^{W} DS2$ (DS7), $DS1 \oplus_{t_0}^{W} DS3$ (DS8). Por ende, para cambios abruptos, $t_0 = 2500$ y $W = 0$. Para cambios graduales, $t_0 = 2500$ y $W = 500$.

En los algoritmos seleccionados se utilizó la configuración por defecto propuesta por los autores. El número de clasificadores base fue fijado a 10 para todos los algoritmos de ensambles, el cual es la configuración por defecto adoptada por MOA. Con respecto al método de detección de cambios de FASE, el tamaño de la prueba estadística para el nivel de alerta fue fijado a $\delta =$ 0,005, y el nivel de cambio fue fijado a $\delta = 0{,}001$. FASE utilizó Naive Bayes como el clasificador base y meta-clasificador en todos los experimentos. Se incluyó además el algoritmo Naive Bayes en el estudio experimental, porque es uno de los algoritmos más usados en la detección de spam.

El rendimiento predictivo de los algoritmos bajo consideración fue evaluado utilizando un corpus de prueba. El problema del cambio de concepto puede ser adverso en las situaciones del mundo real, como es el caso de los correos spam. Para los conjuntos de datos construidos, no hay una noción firme sobre la presencia o tipo de cambio. Los algoritmos fueron evaluados

Tabla 2.2: Desempeño predictivo de los algoritmos.

(a) Desempeño de los algoritmos con 500 atributos.

Algoritmo	FASE	LeveragingBag	NaiveBayes	OzaBagAdwin
DS1	**99,91 ±0,54**	99,40 ±4,07	94,88 ±11,54	97,00 ±11,17
DS2	**99,02 ±3,64**	98,69 ±5,21	91,50 ±11,14	98,04 ±9,85
DS3	**99,92 ±0,53**	99,39 ±4,14	92,29 ±10,73	98,34 ±8,83
DS4	**99,91 ±0,54**	99,43 ±3,98	91,61 ±10,58	97,73 ±11,87
DS5	**99,93 ±0,49**	99,54 ±3,59	94,12 ±11,09	96,65 ±12,55
DS6	**99,44 ±3,09**	99,17 ±3,48	95,54 ±11,09	97,10 ±12,08

(b) Desempeño de los algoritmos con 600 atributos.

Algoritmo	FASE	LeveragingBag	NaiveBayes	OzaBagAdwin
DS1	**99,91 ±0,54**	99,39 ±4,20	94,09 ±12,61	96,58 ±11,81
DS2	**98,98 ±2,98**	98,44 ±5,90	90,29 ±12,32	97,77 ±10,69
DS3	**99,92 ±0,53**	99,39 ±4,14	91,08 ±11,85	98,29 ±9,06
DS4	**99,91 ±0,54**	99,43 ±3,98	89,79 ±11,96	97,64 ±12,31
DS5	**99,93 ±0,49**	99,55 ±3,47	93,43 ±12,23	96,19 ±12,94
DS6	**99,46 ±2,89**	99,15 ±3,95	95,10 ±11,89	96,60 ±13,05

procesando las instancias en su orden temporal y el método *test-then-train* fue adoptado para calcular la precisión predictiva. La Tabla 2.2 muestra el desempeño predictivo de los algoritmos para los seis conjuntos de datos creados. Los niveles más altos del desempeño productivo están resaltados. La Tabla 2.2 muestra que FASE supera al resto de los algoritmos con respecto a la precisión predictiva. Además, todos los algoritmos obtuvieron buenos resultados con 500 y 600 atributos. Adicionalmente, la Tabla 2.3 muestra la precisión y recuperación de los algoritmos para los conjuntos de datos DS7 y DS8 para cambios abruptos y graduales. De acuerdo con la Figura 2.3, FASE y LeveragingBag son capaces de estabilizar el aprendizaje cuando los conceptos son estables. La Figura 2.3 muestra también que los algoritmos que tienen mecanismos de detección de cambio a menudo adaptan el aprendizaje más efectivamente que Naive Bayes.

La Figura 2.4 muestra la jerarquía de los algoritmos con respecto a la Tabla 2.2. Para verificar las diferencias significativas, se utilizó la prueba de Friedman y el procedimiento de Holm para el análisis a posteriori. Los grupos de clasificadores que no son significativamente diferentes (con $p = 0{,}05$) están conectados. Esta figura muestra que, en general, FASE y LeveragingBag se posicionan mejor que el resto de los algoritmos.

Tabla 2.3: Precisión y *recall* de los algoritmos.

(a) Precisión y *recall* de los algoritmos con 500 atributos.

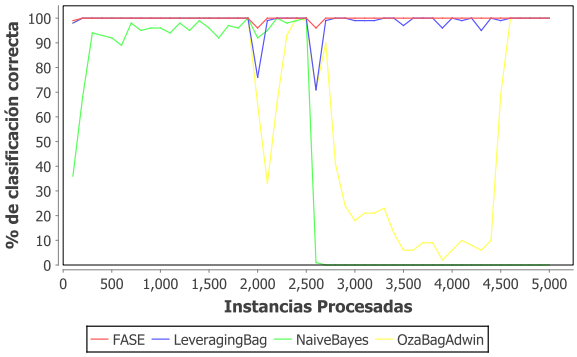
Algoritmo	Medida	DS1	DS2	DS3	DS4	DS5	DS6
FASE	precision	**0.99**	0.97	**0.99**	**0.99**	**0.99**	**0.99**
	recall	0.99	0.99	0.99	0.99	0.99	0.99
LeveragingBag	precision	0.98	**0.99**	**0.99**	**0.99**	**0.99**	0.98
	recall	0.99	0.96	0.98	0.98	0.99	0.99
NaiveBayes	precision	**0.99**	0.77	0.78	0.77	0.96	**0.99**
	recall	0.86	0.99	0.98	0.99	0.86	0.88
OzaBagAdwin	precision	0.92	0.99	0.99	0.99	0.91	0.93
	recall	0.99	0.93	0.94	0.92	0.99	0.99

(b) Precisión y *recall* de los algoritmos con 600 atributos.

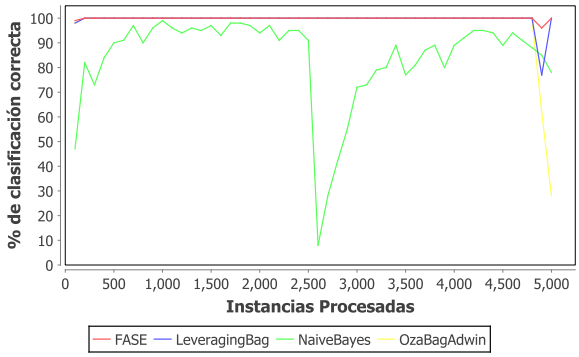
Algoritmo	Medida	DS1	DS2	DS3	DS4	DS5	DS6
FASE	precision	**0.99**	0.97	**0.99**	**0.99**	**0.99**	**0.99**
	recall	0.99	0.99	0.99	0.99	0.99	0.99
LeveragingBag	precision	0.98	**0.99**	**0.99**	**0.99**	**0.99**	0.98
	recall	0.99	0.95	0.98	0.98	0.99	0.99
NaiveBayes	precision	**0.99**	0.75	0.75	0.73	0.97	**0.99**
	recall	0.84	0.99	0.98	0.99	0.84	0.87
OzaBagAdwin	precision	0.91	**0.99**	**0.99**	**0.99**	0.90	0.92
	recall	0.99	0.92	0.93	0.91	0.99	0.99

2.4. Conclusiones parciales

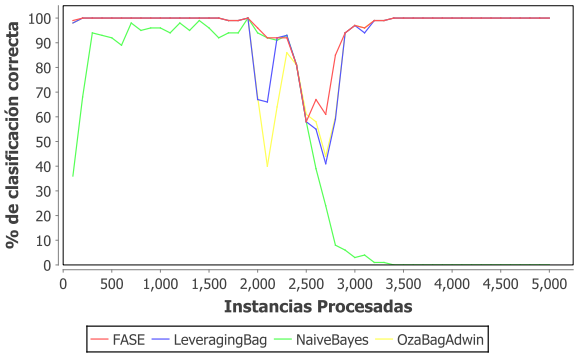
El nuevo modelo para la detección de spam consta de dos etapas, una para el entrenamiento de los clasificadores incrementales y otra para la clasificación de spam, siguiendo un procedimiento general estructurado en tres etapas con sus fases correspondientes. En la fase experimental del modelo, se evaluaron tres algoritmos de ensamble incrementales capaces de manejar el cambio de concepto. Estos algoritmos no han sido utilizados previamente en el contexto de la detección de correos spam. Específicamente, se incluye en el estudio los algoritmos FASE, LeveragingBag y OzaBagAdwin. También se incluye el Naive Bayes porque es uno de los algoritmos más eficaces para aprender de los flujos de datos. En los experimentos se mostró que los mejores algoritmos fueron FASE y LeveragingBag. Además, los algoritmos fueron evaluados en presencia del cambio de concepto. En este caso el peor desempeño fue el de Naive Bayes, ya que no tiene



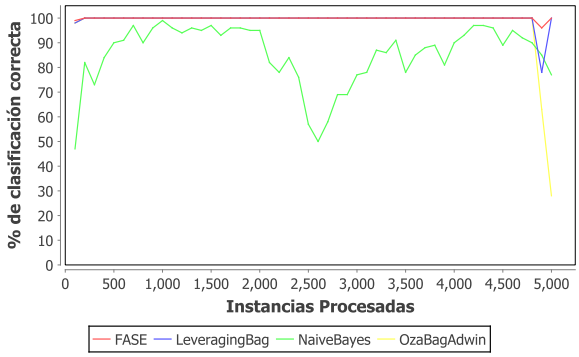
(a) Cambios abruptos en DS7.



(b) Cambios abruptos en DS8.



(c) Cambios graduales en DS7.



(d) Cambios graduales en DS8.

Figura 2.3: Rendimiento predictivo de los algoritmos con cambios abruptos y graduales.

un mecanismo para adaptarse a los cambios de concepto. Los resultados experimentales de los algoritmos seleccionados muestra que las técnicas de minería de datos en combinación con los algoritmos de aprendizaje incremental son una buena opción para la detección de correos spam. Se planea continuar con esta investigación utilizando otros algoritmos de aprendizaje y otros métodos para la selección de atributos.

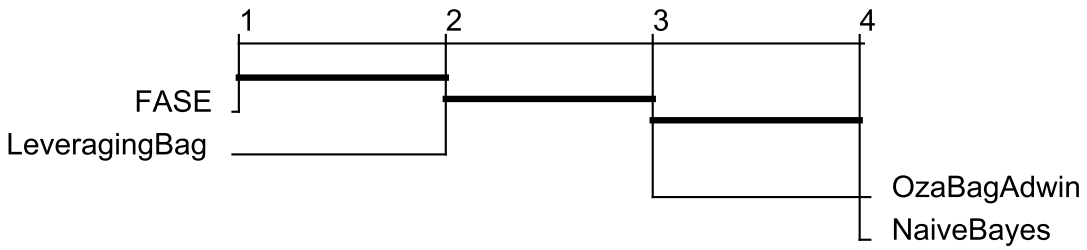


Figura 2.4: Comparación de todos los clasificadores con la prueba de Friedman y el procedimiento de Holm para el análisis a posteriori. Los rangos fueron calculados en consecuencia con las Tablas 2.2a y 2.2b.

CAPÍTULO 3

Prototipo para la detección personalizada de spam aplicando clasificadores incrementales

Las aplicaciones reales de la minería de textos son de especial interés para el aprendizaje automático y la comunidad científica (Chan et al., 2015; Guzella y Caminhas, 2009; Kavanagh, 2012; Li et al., 2017; Santos et al., 2012), debido a que éstas introducen varios retos. Estas aplicaciones tienen que lidiar constantemente con grandes volúmenes de datos (generalmente no estructurados) y debido a la dinámica de muchas situaciones reales, también puede existir cambio de concepto. En este capítulo se propone una aplicación capaz de filtrar correos spam, así como realizar otras tareas comunes en un cliente de correo.

3.1. Descripción general de la herramienta

Para su rápido desarrollo, esta aplicación se apoyó en proyectos y bibliotecas libres y de código abierto disponibles en repositorios como PyPi[1] y GitHub[2], mantenidos por la comunidad de programadores, lo cual le aportó a la aplicación una base de código sólida con un mínimo esfuerzo. Como se muestra en la figura 3.1, se utilizó el lenguaje de programación Python para la creación del preprocesador, por ser un lenguaje multiplataforma, y por la gran disponibilidad de bibliotecas para la minería de datos y el procesamiento de texto que existen para este lengua-

[1] https://pypi.org/
[2] http://github.com/

je, como lo son la biblioteca NLTK (Natural Language ToolKit) la cual ofrece facilidades para la lematización, *stemming*, eliminación de *stop-words* y tokenización en general, y la biblioteca Scikit-learn para la extracción de atributos y el cálculo de medidas como *tf* o *tf-idf*, así como para la reducción de dimensionalidad. También se utilizó Python para la interacción con el servidor de correos ya que bibliotecas para el manejo de los protocolos relacionados con el correo electrónico vienen incluidas por defecto en el lenguaje, así como para integrar la interfaz gráfica con el resto de los módulos de la aplicación. Se utilizó el lenguaje Java para el clasificador por ser un lenguaje multiplataforma y además es el lenguaje en el que están implementadas herramientas fundamentales en el área del aprendizaje automático como Weka[3] y MOA[4], las cuales proveen la implementación de una gran cantidad de algoritmos para la clasificación, así como herramientas para evaluar el desempeño de dichos algoritmos. Para la interfaz gráfica se utilizaron los lenguajes PHP, HTML5, CSS3 y Java Script ya que se utilizó el código fuente de la herramienta Rainloop[5] la cual es un cliente de correos simple, rápido y moderno basado en tecnologías web.

[3] https://www.cs.waikato.ac.nz/ml/weka/
[4] https://moa.cms.waikato.ac.nz/
[5] https://www.rainloop.net/

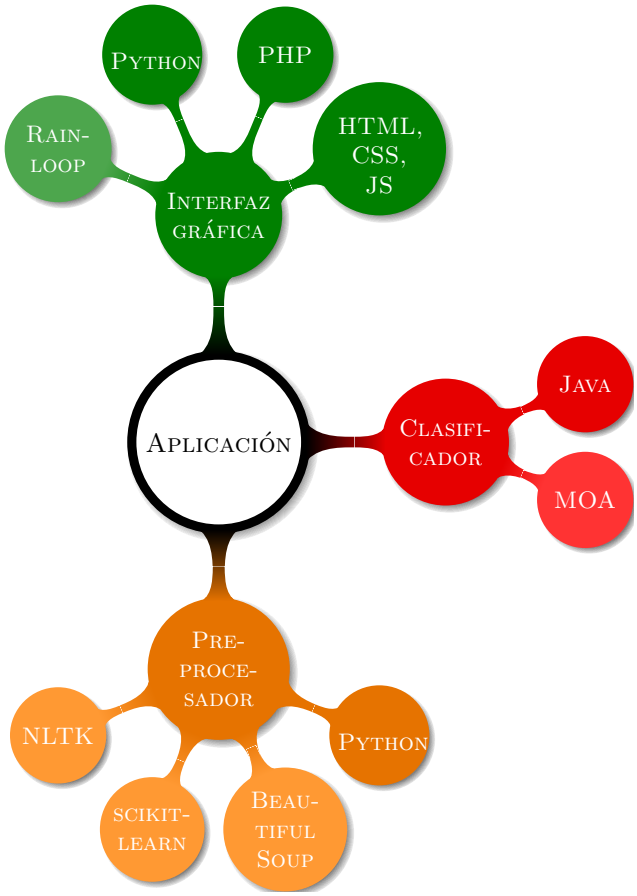
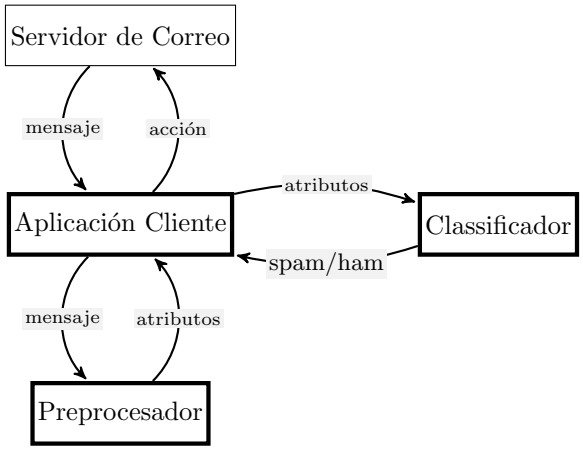


Figura 3.1: Herramientas y lenguajes de programación utilizados en la aplicación.

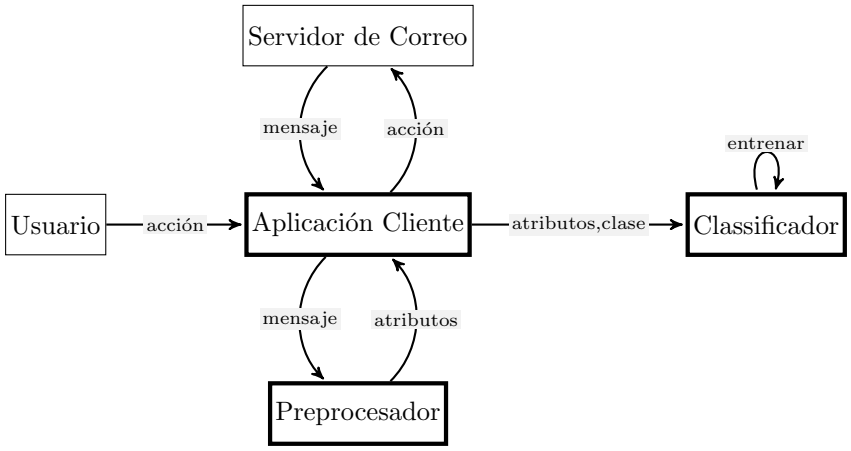
La aplicación realiza dos funciones fundamentales (además de las funciones que comúnmente están disponibles en todo cliente de correos, como mover, eliminar, visualizar, redactar y enviar mensajes):

1. **Filtrar spam (Figura 3.2a):** En esta fase la aplicación toma los mensajes que van arribando al servidor de correo, los manda a preprocesar con el módulo de preprocesamiento, para luego enviar los mensajes preprocesados al módulo de clasificación, el cual asigna una etiqueta de clase (ham o spam) al mensaje, con esta etiqueta la aplicación entonces decide si debe marcar el mensaje como spam en el servidor.

2. **Aprender de los errores (Figura 3.2b):** En esta fase la aplicación detecta, a través de las acciones del usuario, que un mensaje ha sido etiquetado erróneamente por el clasificador, por lo que toma este mensaje del servidor lo envía al preprocesador y envía al clasificador, para su entrenamiento, el mensaje preprocesado y la clase correcta a la que pertenece.



(a) Fase de clasificación.



(b) Fase de aprendizaje.

Figura 3.2: Funcionamiento de la herramienta.

3.2. Módulos principales

La aplicación está compuesta por tres módulos completamente independientes e intercambiables (incluso en tiempo de ejecución de la aplicación):

1. **Módulo de preprocesamiento:** Es un servicio cuya única tarea es recibir un correo electrónico en bruto y retornar una representación de bolsa de palabras del mismo.

2. **Módulo de aprendizaje:** Es un servicio que contiene un clasificador, su función es la de dado un mensaje preprocesado retornar la clase a la que el clasificador predice que

pertenece, o dado un mensaje y la clase a la que pertenece, realizar el entrenamiento del clasificador para mejorar futuras predicciones.

3. **Módulo de interfaz de usuario:** Es el que permite la interacción del usuario con el resto de la aplicación. Es el encargado de interactuar con el servidor de correo, enviar los mensajes al preprocesador y los mensajes preprocesados al clasificador así como realizar acciones con los mensajes en el servidor de correo, basadas en las predicciones del clasificador.

3.3. Interfaz de usuarios

En la figura 3.3 se muestra la ventana de inicio de la aplicación, la cual está compuesta por un formulario que el usuario debe completar con su nombre de usuario y clave, para poder ingresar a su bandeja de entrada. También dispone de un botón (en forma de globo terráqueo) para cambiar el idioma de la interfaz de la aplicación como se muestra en la figura 3.4. Una vez iniciada la sesión (Figura 3.5) el programa le muestra al usuario su bandeja de entrada, así como otras carpetas (de existir), al mismo tiempo el programa en el trasfondo está comprobando si hay mensajes nuevos y moviéndolos a la carpeta de spam de haber sido etiquetados como tal por el clasificador, o aprendiendo de los mensajes que clasificó incorrectamente, mediante las acciones del usuario. La interfaz de usuario trata de ser moderna, sencilla y lo más amena posible, pero si el usuario no se siente a gusto con la apariencia del programa puede ir a la sección de configuración (Figura 3.6) y escoger entre una variedad de temas el que vaya mejor con su estilo, así como cambiar otros aspectos configurables como el idioma, crear carpetas, etc.

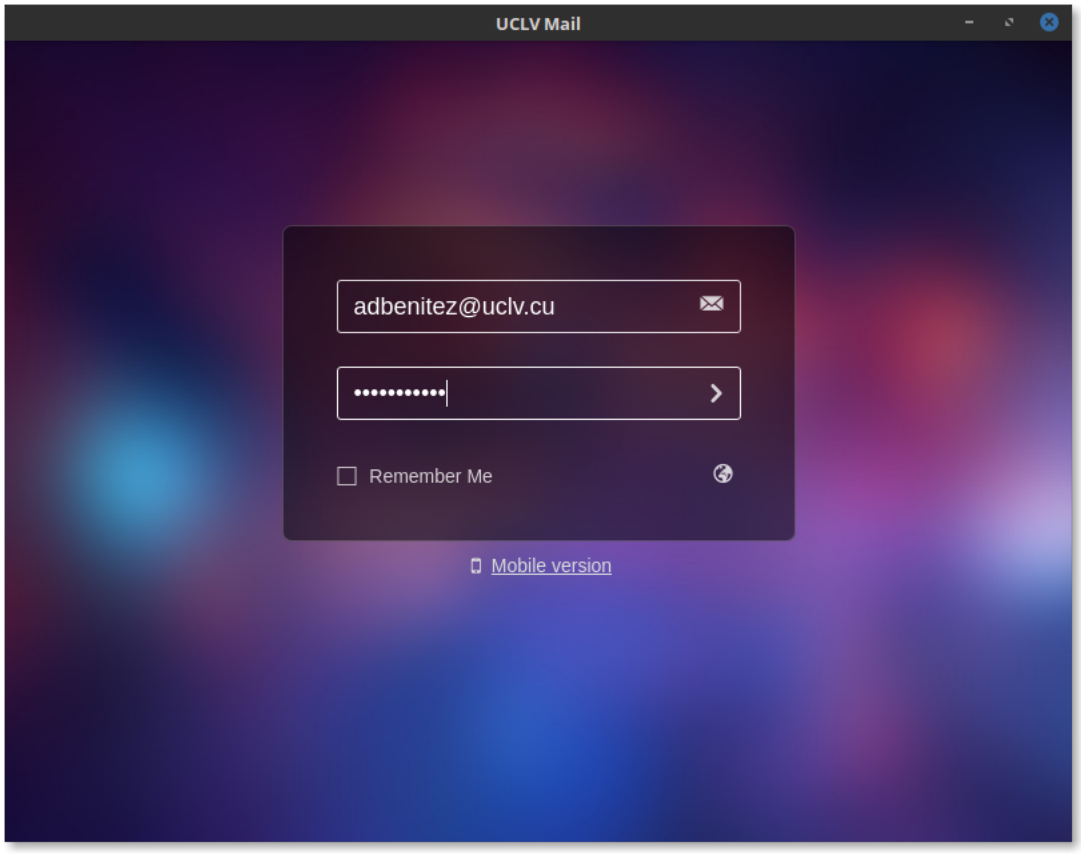


Figura 3.3: Formulario de inicio de sesión.

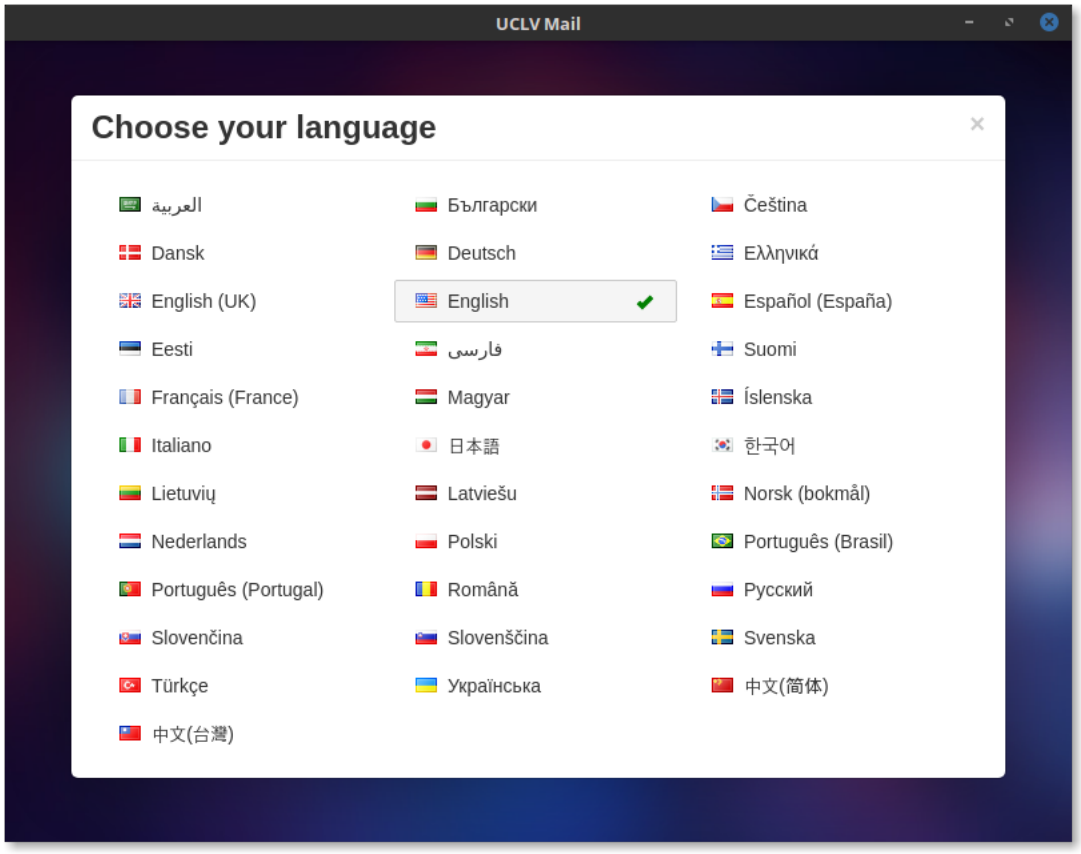


Figura 3.4: Selección de idioma.

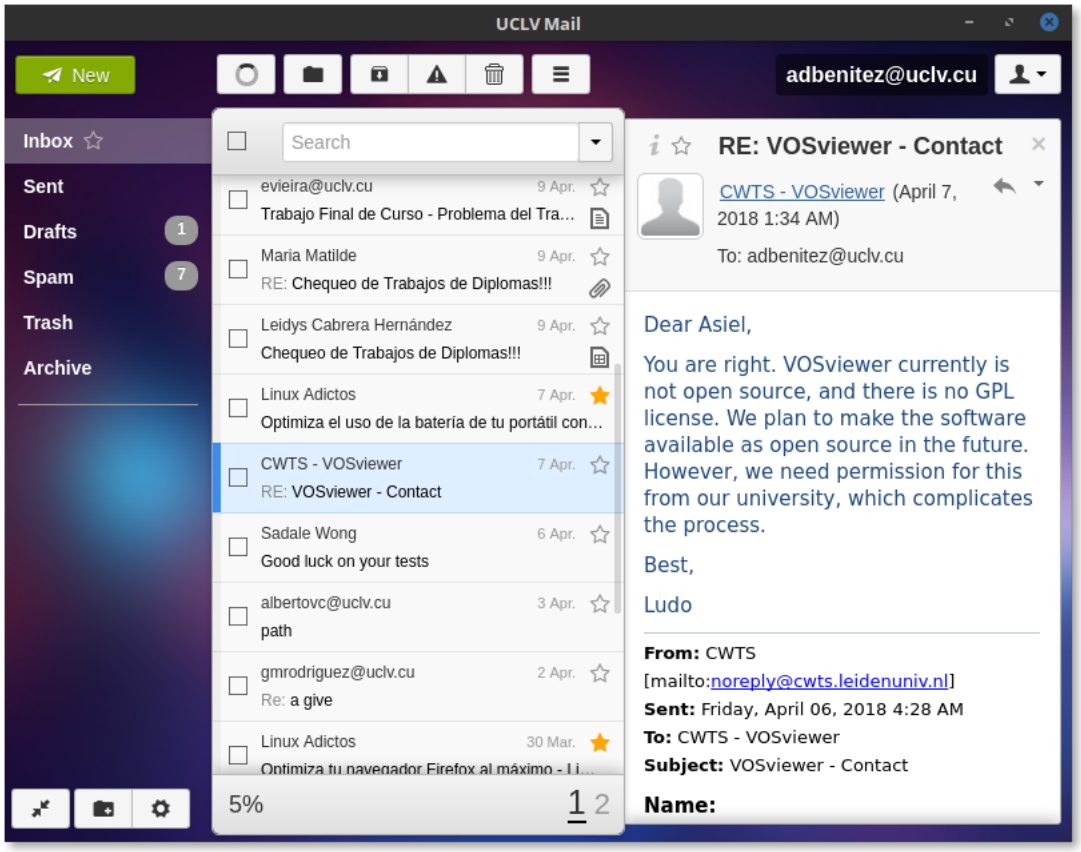


Figura 3.5: Bandeja de entrada.

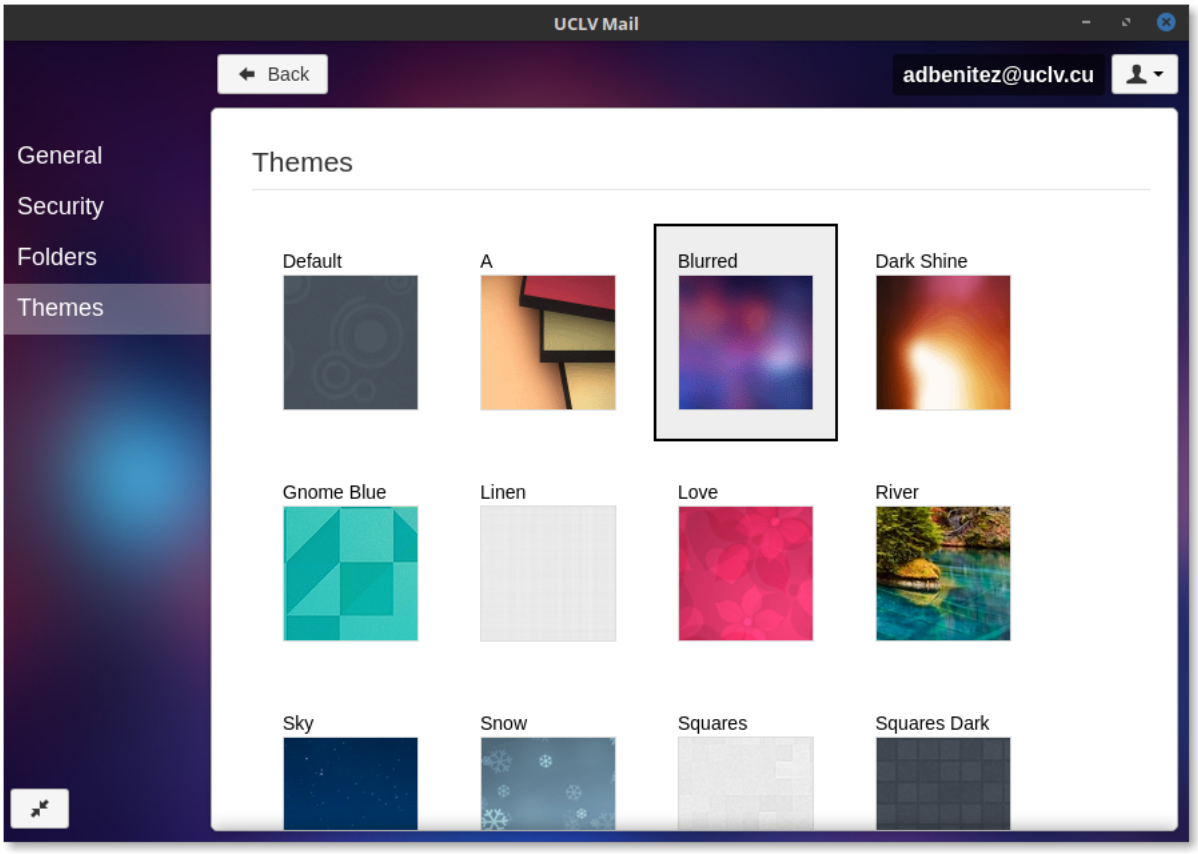


Figura 3.6: Ventana de configuración.

3.4. Conclusiones parciales

Python es un lenguaje muy popular para el procesamiento del lenguaje natural y la minería de textos. El Weka y el MOA son herramientas populares para el entrenamiento y validación de algoritmos de aprendizaje automático. Se creó una aplicación cliente de correo que además de ofrecer las funcionalidades básicas esperadas de todo cliente de correo, es capaz (usando clasificadores incrementales) de filtrar spam y aprender de sus errores, o sea, que es entrenado automáticamente a través de las acciones del usuario, logrando reconocer el spam cada vez con mayor precisión, así como adaptarse a los cambios de concepto presentes en un flujo de mensajes cambiante en el tiempo.

Conclusiones

Con el desarrollo de esta investigación se logró detectar de manera eficiente y eficaz los correos spam mediante el diseño de un modelo que combina armónicamente técnicas de minería de texto y clasificadores incrementales, cumpliéndose de esta forma el objetivo general propuesto, ya que:

- Los correos electrónicos son infinitos en el tiempo, por lo que resulta apropiada la utilización de los algoritmos incrementales en lugar de algoritmos de clasificación por lotes para la detección de spam.
- El nuevo modelo para la detección de spam consta de dos etapas, una para el entrenamiento de los clasificadores incrementales y otra para la clasificación de spam, siguiendo un procedimiento general estructurado en tres etapas con sus fases correspondientes.
- De los tres algoritmos capaces de manejar el cambio de concepto, los de mejores resultados fueron FASE y LeveragingBag. Por otro lado, Naive Bayes obtuvo los peores resultados por no tener un mecanismo de detección explícita de cambio de concepto.
- La aplicación cliente de correo desarrollada, además de ofrecer las funcionalidades básicas de todo cliente, es capaz de filtrar los correos spam, retroalimentarse del usuario y adaptarse a los cambios de concepto presentes en los flujos de mensajes cambiantes en el tiempo, para reconocer el spam cada vez con mayor precisión.

Recomendaciones

A pesar de todo el trabajo realizado en la presente investigación, aún quedan temas que no se abordaron y que pueden hacer la detección de spam aun más efectiva, por lo que se recomienda:

- Extender la aplicación para que sea posible preprocesar mensajes en idioma Español ya que actualmente el programa solo es capaz de preprocesar correctamente mensajes en idioma Inglés, debido a que usa el WordNet para lematizar los mensajes.
- Usar un detector de idioma como la biblioteca langdetect[6] para preprocesar los mensajes en dependencia del idioma.
- Utilizar un corrector ortográfico como la biblioteca autocorrect[7] para evitar que palabras con errores ortográficos sean tratadas como una palabra distinta de la que se intentó expresar.
- Realizar una selección incremental de atributos. Esto permitiría al clasificador aprender nuevos atributos que no existen en el corpus inicial con que fue creado y entrenado.

[6] https://pypi.org/project/langdetect/
[7] https://pypi.org/project/autocorrect/

REFERENCIAS

Amidon, A. (2017). A New Approach Adapting Neural Network Classifiers to Sudden Changes in Nonstationary Environments.

Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C. D. y Stamatopoulos, P. (2000). Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach, *arXiv preprint cs/0009009* .

Arco García, L. (2005). *Modelo para el agrupamiento de documentos afines y su ulterior resumen a través de la representación espacio vectorial de un corpus textual*, Tesis de maestría, Universidad Central "Marta Abreu" de Las Villas, Santa Clara, Cuba.

Baena-Garcia, M., Campo-Avila, J. d., Fidalgo, R., Bifet, A., Gavalda, R. y Morales-Bueno, R. (2006). Early drift detection method.

Bifet, A. (2009). *Adaptive learning and mining for data streams and frequent patterns*, PhD thesis.

Bifet, A. y Frank, E. (2010). Sentiment knowledge discovery in twitter streaming data, *International Conference on Discovery Science*, Springer, pp. 1–15.

Bifet, A. y Gavalda, R. (2007). Learning from time-changing data with adaptive windowing, *In SIAM International Conference on Data Mining*.

Bifet, A., Holmes, G., Kirkby, R. y Pfahringer, B. (2010). Moa: Massive online analysis, *The Journal of Machine Learning Research* **11**: 1601–1604.

Bifet, A., Holmes, G. y Pfahringer, B. (2010). Leveraging bagging for evolving data streams, *Machine learning and knowledge discovery in databases*, Springer, pp. 135–150.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R. y Gavalda, R. (2009). New ensemble methods for evolving data streams, *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 139–148.

Blanco, I. F. (2014). *Nuevos métodos para el aprendizaje en flujos de datos no estacionarios*, PhD thesis, Universidad de Granada.

Breiman, L. (1996). Bagging predictors, *Machine learning* **24**(2): 123–140.

Breiman, L., Friedman, J., Stone, C. J. y Olshen, R. A. (1984). *Classification and regression trees*, CRC press.

Brzezinski, D. y Stefanowski, J. (2014). Combining block-based and online methods in learning ensembles from concept drifting data streams, *Information Sciences* **265**: 50 – 67.

Chan, P. P., Yang, C., Yeung, D. S. y Ng, W. W. (2015). Spam filtering for short messages in adversarial environment, *Neurocomputing* **155**: 167–176.

Del Campo Ávila, J. (2007). Nuevos enfoques en aprendizaje incremental.

Domingos, P. y Hulten, G. (2000). Mining high-speed data streams, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 71–80.

Ferrer, F. J., Aguilar, J. S. y Riquelme, J. C. (2005). Aprendizaje Incremental de Reglas en Data Streams, *Actas del III Taller Nacional de Minería de Datos y Aprendizaje, TAMIDA2005* pp. 261–270.

Ferrer-Troyano, F., Aguilar-Ruiz, J. S. y Riquelme, J. C. (2004). Discovering decision rules from numerical data streams, *Proceedings of the 2004 ACM symposium on Applied computing*, ACM, pp. 649–653.

Freund, Y., Schapire, R. E. y others (1996). Experiments with a new boosting algorithm, *ICML*, Vol. 96, pp. 148–156.

Frias-Blanco, I., Campo-Avila, J. d., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A. y Caballero-Mota, Y. (2015). Online and Non-Parametric Drift Detection Methods Based on Hoeffding Bounds, *IEEE Transactions on Knowledge and Data Engineering* **27**(3): 810–823.

Frías-Blanco, I., del Campo-Ávila, J., Ramos-Jiménez, G., Carvalho, A. C., Ortiz-Díaz, A. y Morales-Bueno, R. (2016). Online adaptive decision trees based on concentration inequalities, *Knowledge-Based Systems* **104**: 179–194.

Frías-Blanco, I., Verdecia-Cabrera, A., Ortiz-Díaz, A. y Carvalho, A. (2016). Fast adaptive stacking of ensembles, *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ACM, pp. 929–934.

Gama, J., Medas, P., Castillo, G. y Rodrigues, P. (2004). Learning with drift detection, *Advances in artificial intelligence*, Springer, pp. 286–295.

Gama, J., Rocha, R. y Medas, P. (2003). Accurate decision trees for mining high-speed data streams, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pp. 523–528.

Gama, J., Rodrigues, P. P. y Sebastião, R. (2009). Evaluating algorithms that learn from data streams, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, ACM, pp. 1496–1500.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M. y Bouchachia, A. (2014). A Survey on Concept Drift Adaptation, *ACM Comput. Surv.* **46**(4): 44:1–44:37.

Geng, X. y Smith-Miles, K. (2009). Incremental Learning, pp. 731–735.

Guzella, T. S. y Caminhas, W. M. (2009). A review of machine learning approaches to Spam filtering, *Expert Systems with Applications* **36**(7): 10206–10222.

He, H. y A. Garcia, E. (2009). Learning from Imbalanced Data, *IEEE Transactions on Knowledge and Data Engineering* **21**(9).

Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables, *Journal of the American Statistical Association* **58**(301): 13–30.

Hulten, G., Spencer, L. y Domingos, P. (2001). Mining Time-changing Data Streams, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, ACM, New York, NY, USA, pp. 97–106.

Idris, I., Selamat, A., Thanh Nguyen, N., Omatu, S., Krejcar, O., Kuca, K. y Penhaker, M. (2015). A combined negative selection algorithm–particle swarm optimization for an email spam detection system, *Engineering Applications of Artificial Intelligence* **39**: 33–44.

Katakis, I., Tsoumakas, G., Banos, E., Bassiliades, N. y Vlahavas, I. (2009). An adaptive personalized news dissemination system, *Journal of Intelligent Information Systems* **32**(2): 191–212.

Kavanagh, R. (2012). Boosting your spam arsenal, *Computer Fraud & Security* **2012**(2): 15–17.

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M. y Ghédira, K. (2016). Discussion and review on evolving data streams and concept drift adapting, *Evolving Systems* pp. 1–23.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J. y Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey, *Information Fusion* **37**: 132–156.

Kubat, M. y Widmer, G. (1995). Adapting to drift in continuous domains, *European Conference on Machine Learning*, Springer, pp. 307–310.

Li, P., He, L., Wang, H., Hu, X., Zhang, Y., Li, L. y Wu, X. (2017). Learning From Short Text Streams With Topic Drifts, *IEEE Transactions on Cybernetics* **PP**(99): 1–15.

Liu, Y. (2004). A Comparative Study on Feature Selection Methods for Drug Discovery, *Journal of Chemical Information and Computer Sciences* **44**(5): 1823–1828.

Liu, Y. y Zhou, Y. (2014). Online Detection of Concept Drift in Visual Tracking, *Neural Information Processing*, Springer, Cham, pp. 159–166.

Losing, V., Hammer, B. y Wersing, H. (2016). KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift.

Metsis, V., Androutsopoulos, I. y Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes?, *CEAS*, Vol. 17, pp. 28–69.

Minku, L. L., White, A. P. y Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift, *Knowledge and Data Engineering, IEEE Transactions on* **22**(5): 730–742.

Mitchell, T. M. (1997). *Machine Learning*, McGraw-Hill series in computer science, McGraw-Hill, New York.

Mladeni'c, D. y Grobelnik, a. M. (1998). Feature Selection for Classification Based on Text Hierarchy, *Text and the Web, Conference on Automated Learning and Discovery CONALD-98*.

Montgomery, D. C. (2007). *Introduction to statistical quality control*, John Wiley & Sons.

Nishida, K. (2008). *Learning and detecting concept drift*, PhD thesis, School of Information Science and Technology, Hokkaido University.

Ortíz Díaz, A., del Campo-Ávila, J., Ramos-Jiménez, G., Frías Blanco, I., Caballero Mota, Y., Mustelier Hechavarría, A. y Morales-Bueno, R. (2014). Fast Adapting Ensemble: A New Algorithm for Mining Data Streams with Concept Drift, *The Scientific World Journal* .

Oza, N. C. y Russell, S. (2001). Online Bagging and Boosting, *in* T. Jaakkola y T. Richardson (eds), *Eighth International Workshop on Artificial Intelligence and Statistics*, Morgan Kaufmann, Key West, Florida. USA, pp. 105–112.

Ozawa, S., Nakazato, J., Ban, T., Shimamura, J. y others (2015). An autonomous online malicious spam email detection system using extended RBF network, *Neural Networks (IJCNN), 2015 International Joint Conference on*, IEEE, pp. 1–7.

Quinlan, J. R. (1986). Induction of decision trees, *Machine learning* **1**(1): 81–106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*, Elsevier.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M. y Herrera, F. (2017). A survey on data preprocessing for data stream mining: Current status and future directions, *Neurocomputing* **239**(C): 39–57.

Rathod, S. B. y Pattewar, T. M. (2015). Content based spam detection in email using Bayesian classifier, *Communications and Signal Processing (ICCSP), 2015 International Conference on*, IEEE, pp. 1257–1261.

Sahami, M., Dumais, S., Heckerman, D. y Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail, *Learning for Text Categorization: Papers from the 1998 workshop*, Vol. 62, pp. 98–105.

Salton, G., Wong, A. y Yang, C. S. (1975). A Vector Space Model for Automatic Indexing, *Commun. ACM* **18**(11): 613–620.

Sanghani, G. y Kotecha, K. (2016). Personalized spam filtering using incremental training of support vector machine, *Computing, Analytics and Security Trends (CAST), International Conference on*, IEEE, pp. 323–328.

Santos, I., Laorden, C., Sanz, B. y Bringas, P. G. (2012). Enhanced Topic-based Vector Space Model for semantics-aware spam filtering, *Expert Systems with Applications* **39**(1): 437–444.

Schlimmer, J. C. y Granger, Jr., R. H. (1986). Incremental Learning from Noisy Data, *Mach. Learn.* **1**(3): 317–354.

Sethi, T. S. y Kantardzic, M. (2017). On the Reliable Detection of Concept Drift from Streaming Unlabeled Data, *Expert Systems with Applications* .

Stanley, K. O. (2003). Learning concept drift with a committee of decision trees, *Informe tecnico: UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA* .

Street, W. N. y Kim, Y. (2001). A Streaming Ensemble Algorithm (SEA) for Large-scale Classification, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, ACM, New York, NY, USA, pp. 377–382.

Sun, Y., Tang, K., Zhu, Z. y Yao, X. (2017). Concept Drift Adaptation by Exploiting Historical Knowledge, *arXiv:1702.03500 [cs]* . arXiv: 1702.03500.

Wang, H., Fan, W., Yu, P. S. y Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 226–235.

Webb, G. I., Lee, L. K., Petitjean, F. y Goethals, B. (2017). Understanding Concept Drift, *arXiv preprint arXiv:1704.00362* .

Widmer, G. y Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts, *Machine learning* **23**(1): 69–101.

Wijaya, A. y Bisri, A. (2016). Hybrid decision tree and logistic regression classifier for email spam detection, *Information Technology and Electrical Engineering (ICITEE), 2016 8th International Conference on*, IEEE, pp. 1–4.

Yang, Y. y Elfayoumy, S. (2007). Anti-Spam Filtering Using Neural Networks and Baysian Classifiers, *2007 International Symposium on Computational Intelligence in Robotics and Automation*, pp. 272–278.

Yang, Y. y Pedersen, J. O. (1997). A comparative study on feature selection in text categorization, *Icml*, Vol. 97, pp. 412–420.

Zhang, Y., Chu, G., Li, P., Hu, X. y Wu, X. (2017). Three-layer Concept Drifting Detection in Text Data Streams, *Neurocomputing* .

Zorkadis, V., Karras, D. y Panayotou, M. (2005). Efficient information theoretic strategies for classifier combination, feature extraction and performance evaluation in improving false positives and false negatives for spam e-mail filtering, *Neural Networks* **18**(5-6): 799–807.

ANEXO A

Tiempo de ejecución de los algoritmos

Tiempo de ejecución de los algoritmos para una selección de 500 atributos

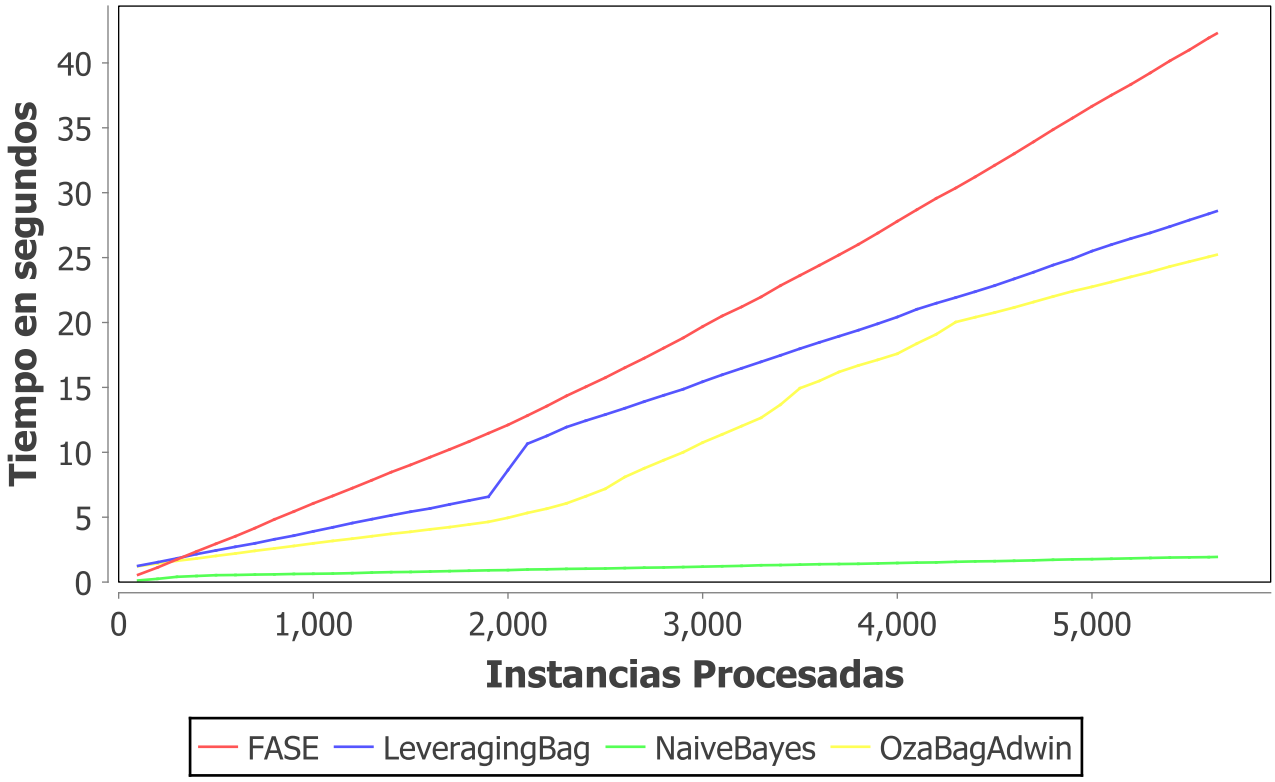


Figura A.1: Tiempo de ejecución de los algoritmos para la colección DS1

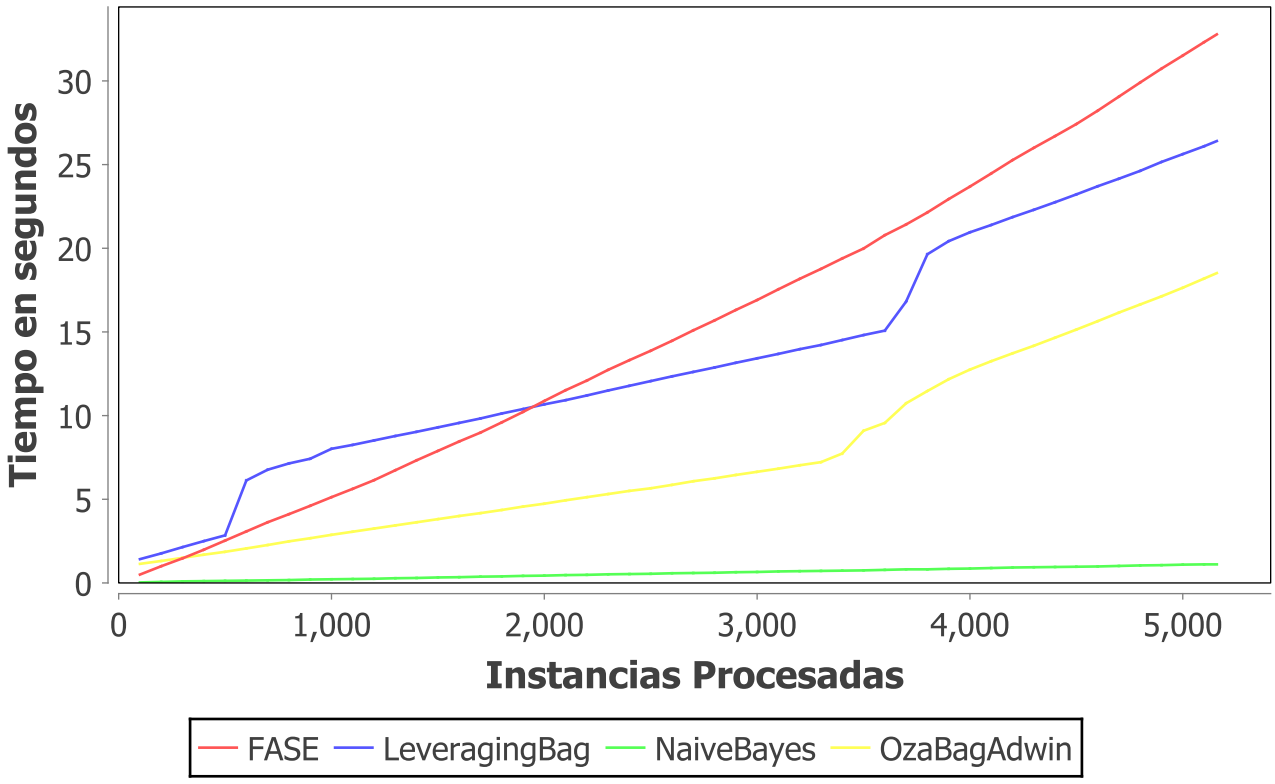


Figura A.2: Tiempo de ejecución de los algoritmos para la colección DS2

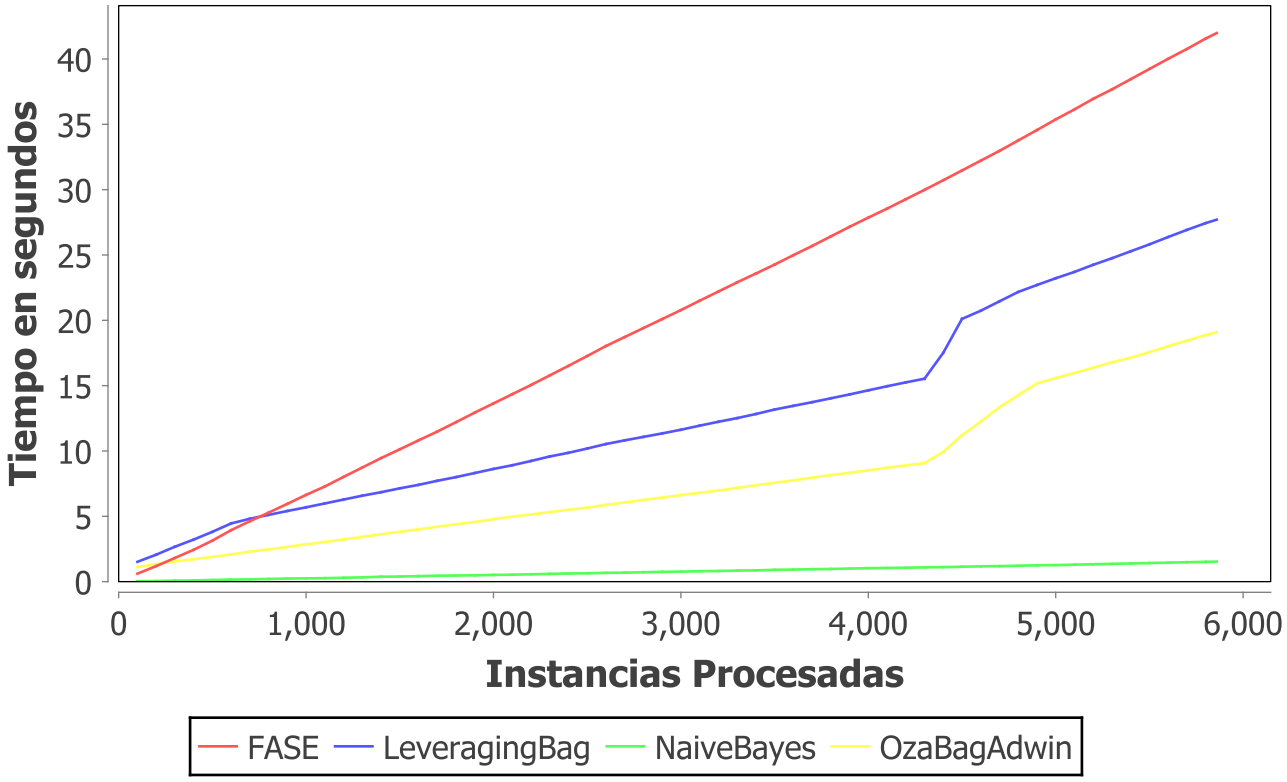


Figura A.3: Tiempo de ejecución de los algoritmos para la colección DS3

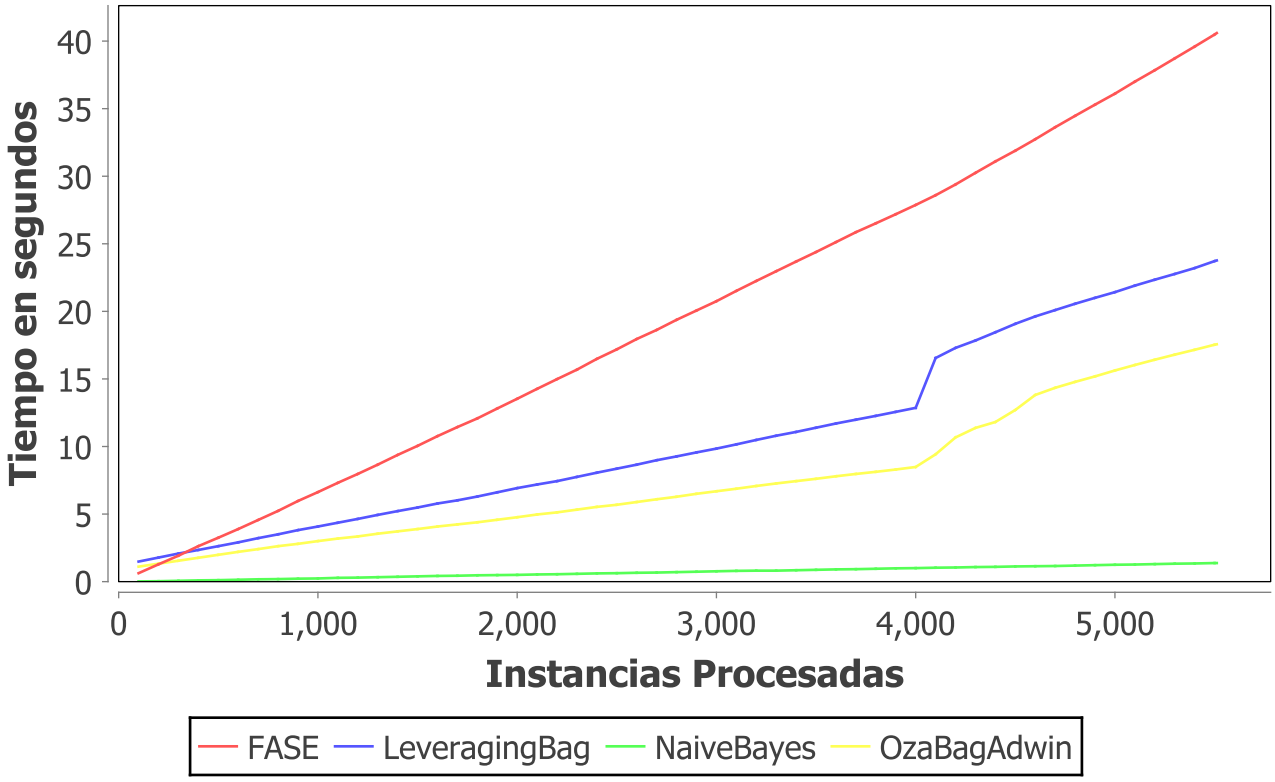


Figura A.4: Tiempo de ejecución de los algoritmos para la colección DS4

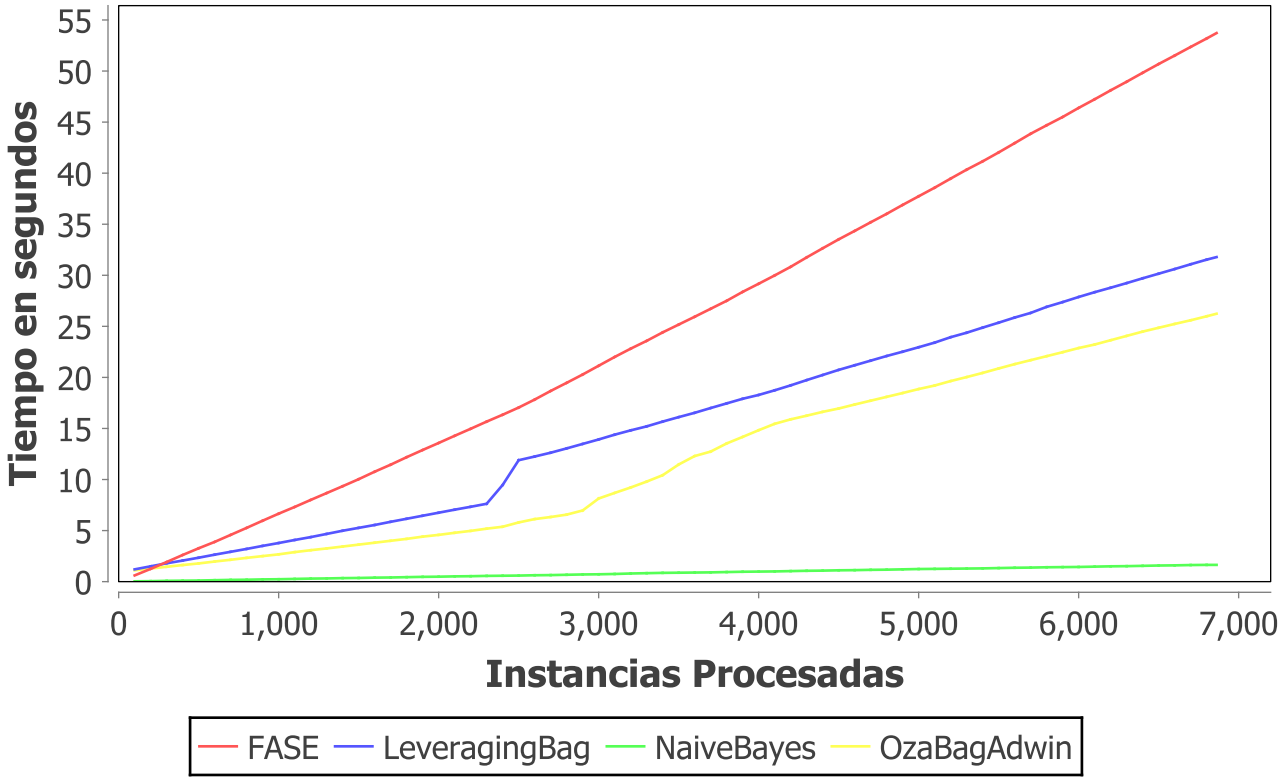


Figura A.5: Tiempo de ejecución de los algoritmos para la colección DS5

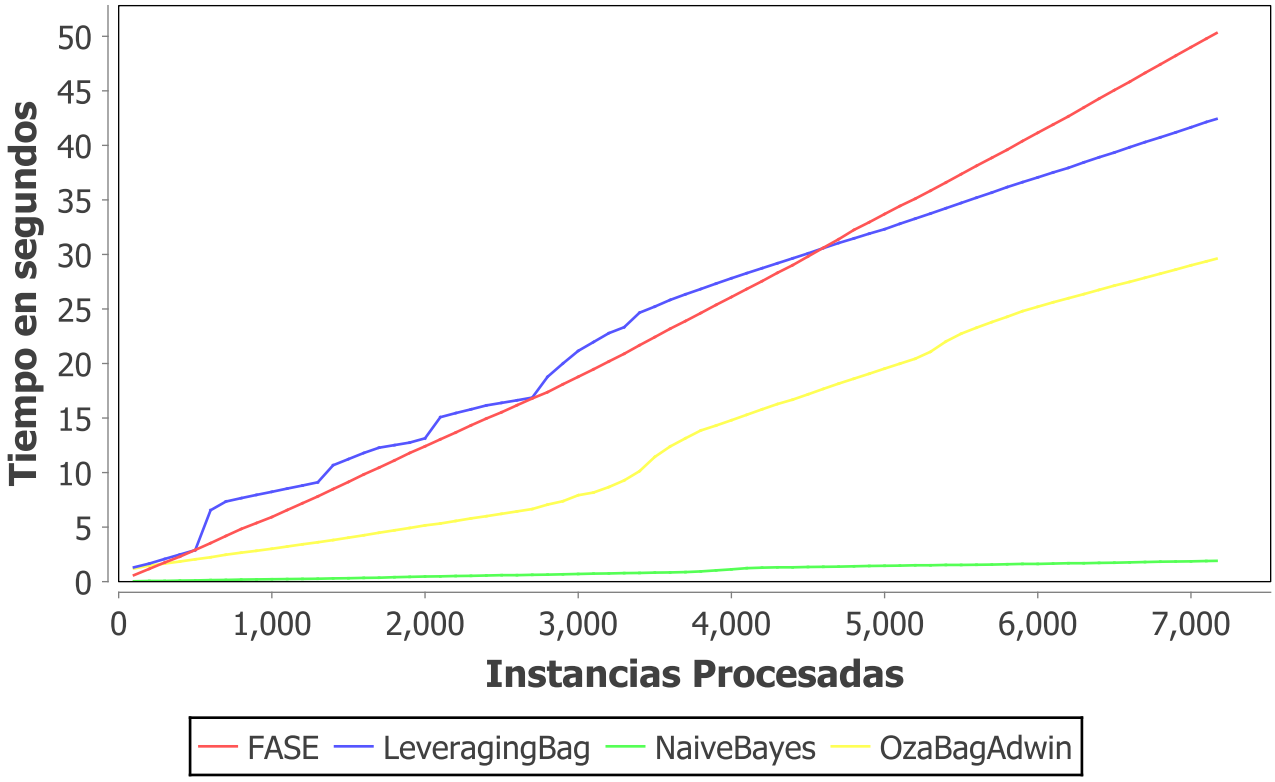


Figura A.6: Tiempo de ejecución de los algoritmos para la colección DS6

Tiempo de ejecución de los algoritmos para una selección de 600 atributos

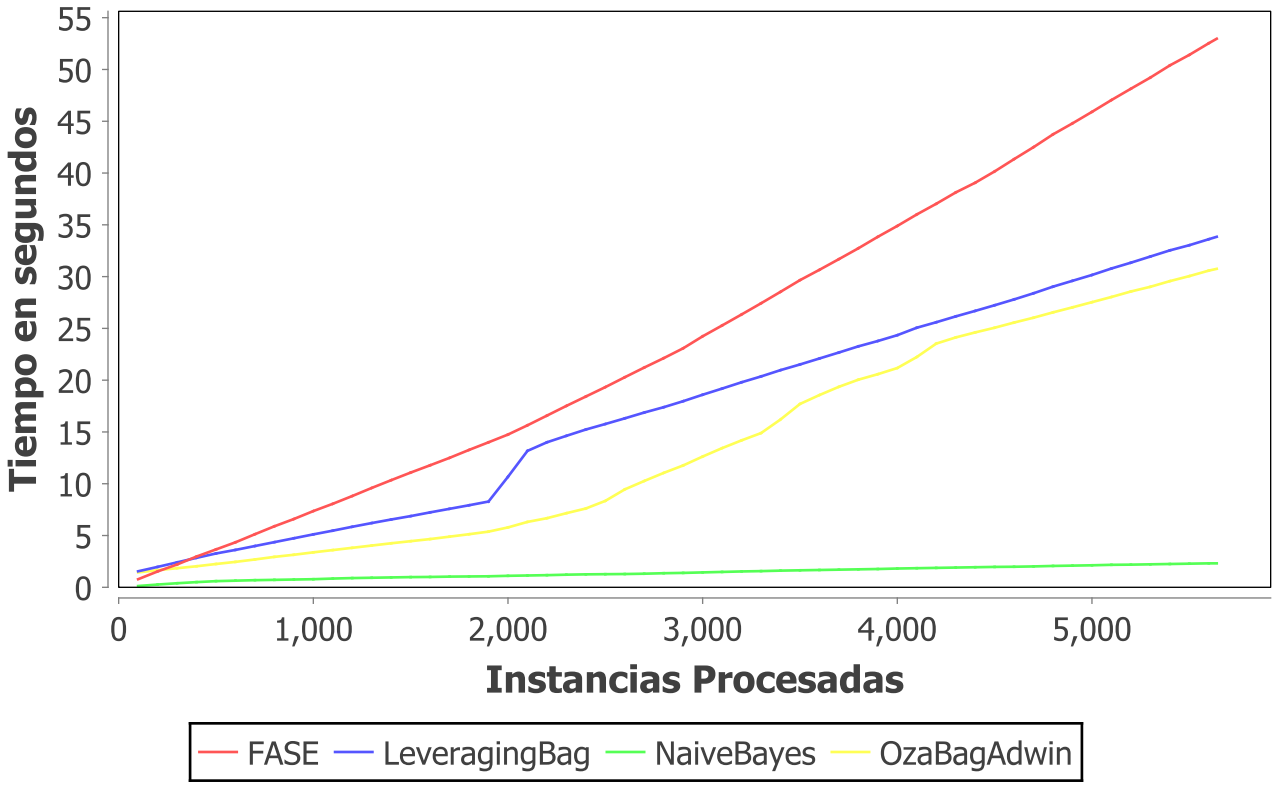


Figura A.7: Tiempo de ejecución de los algoritmos para la colección DS1

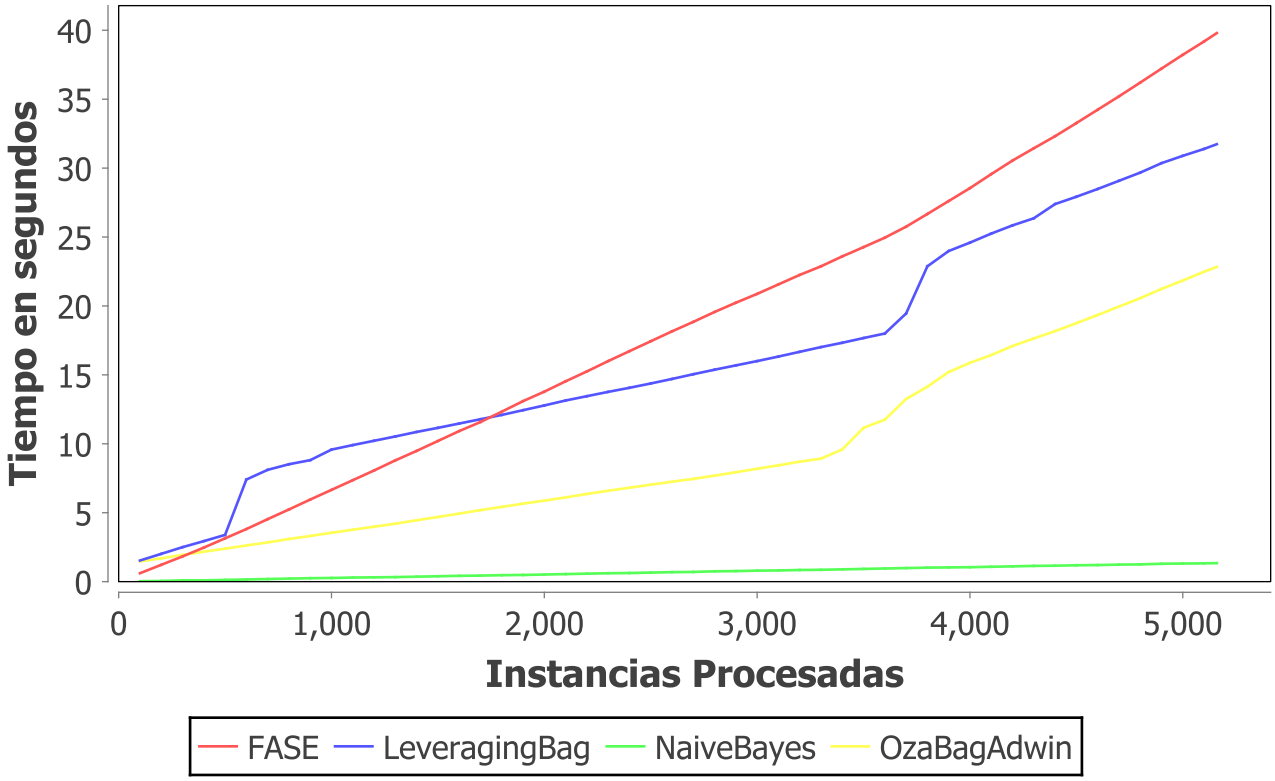


Figura A.8: Tiempo de ejecución de los algoritmos para la colección DS2

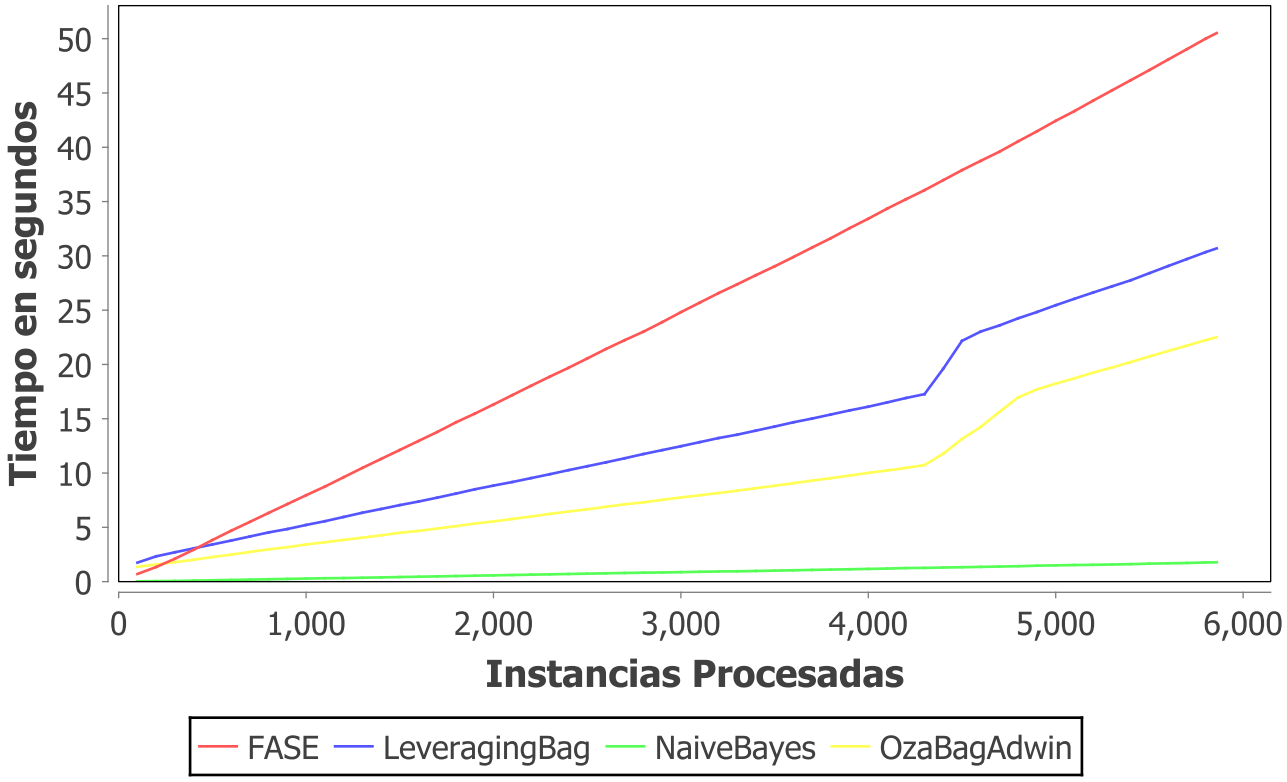


Figura A.9: Tiempo de ejecución de los algoritmos para la colección DS3

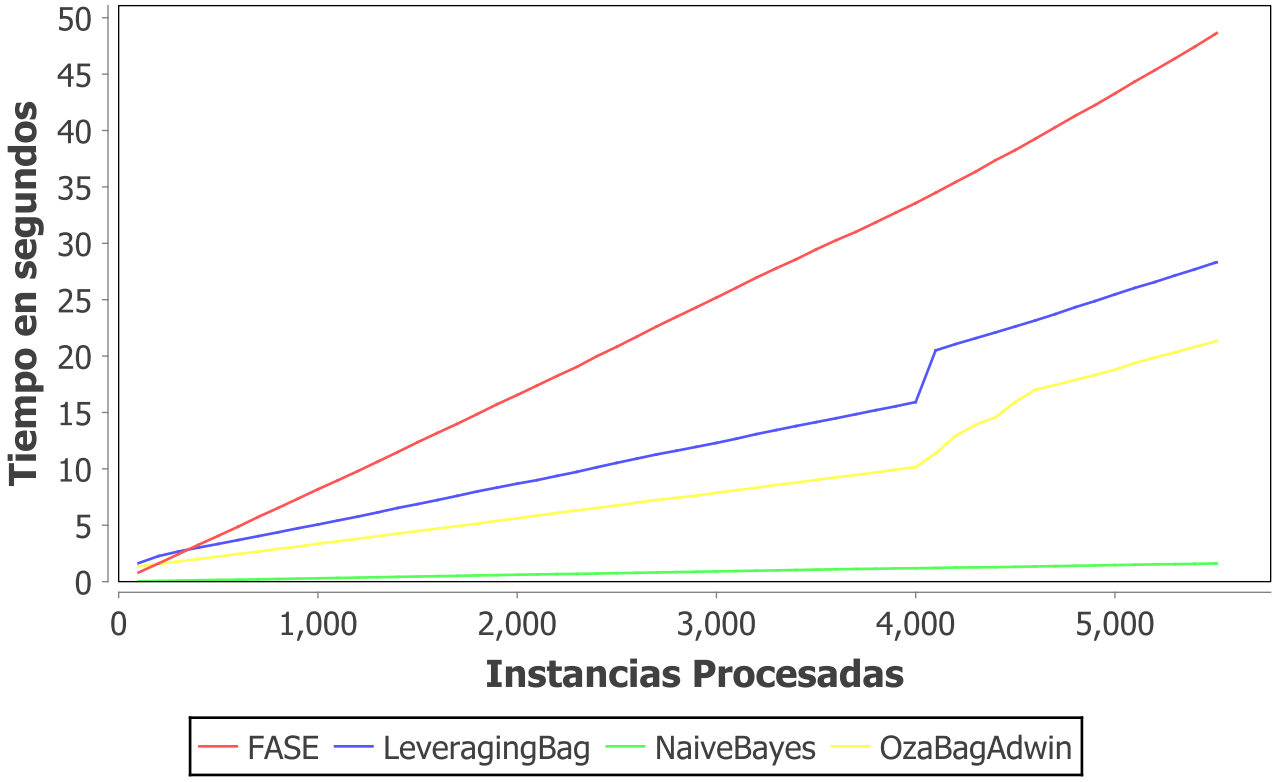


Figura A.10: Tiempo de ejecución de los algoritmos para la colección DS4

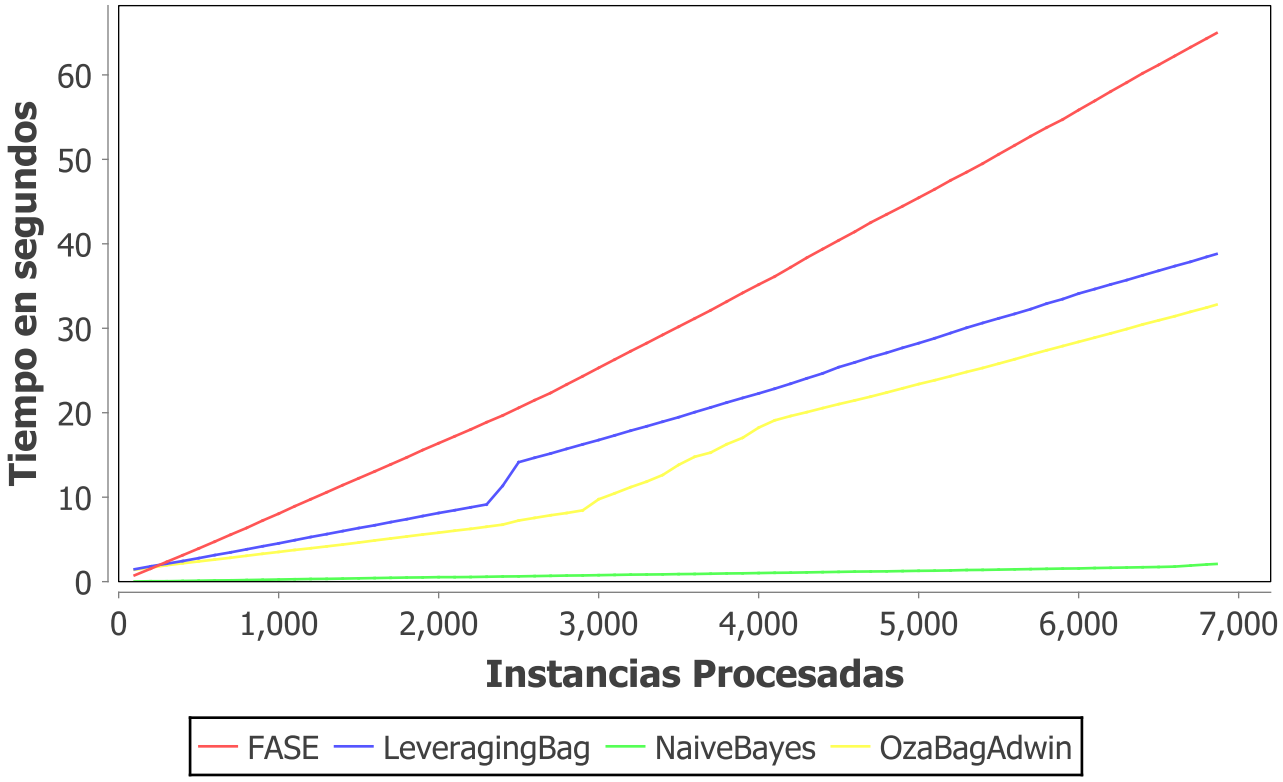


Figura A.11: Tiempo de ejecución de los algoritmos para la colección DS5

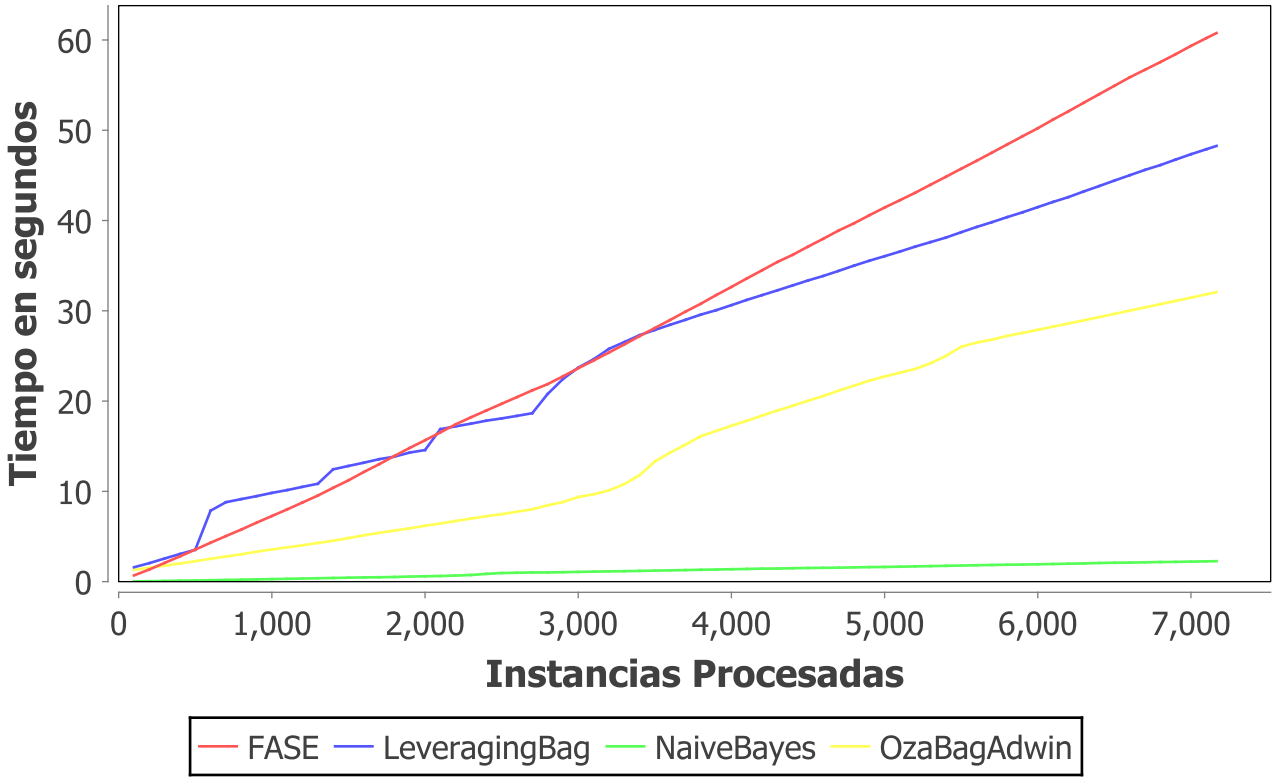


Figura A.12: Tiempo de ejecución de los algoritmos para la colección DS6

ANEXO B

Consumo de memoria de los algoritmos

Consumo de memoria (en MegaBytes) de los algoritmos para una selección de 500 atributos

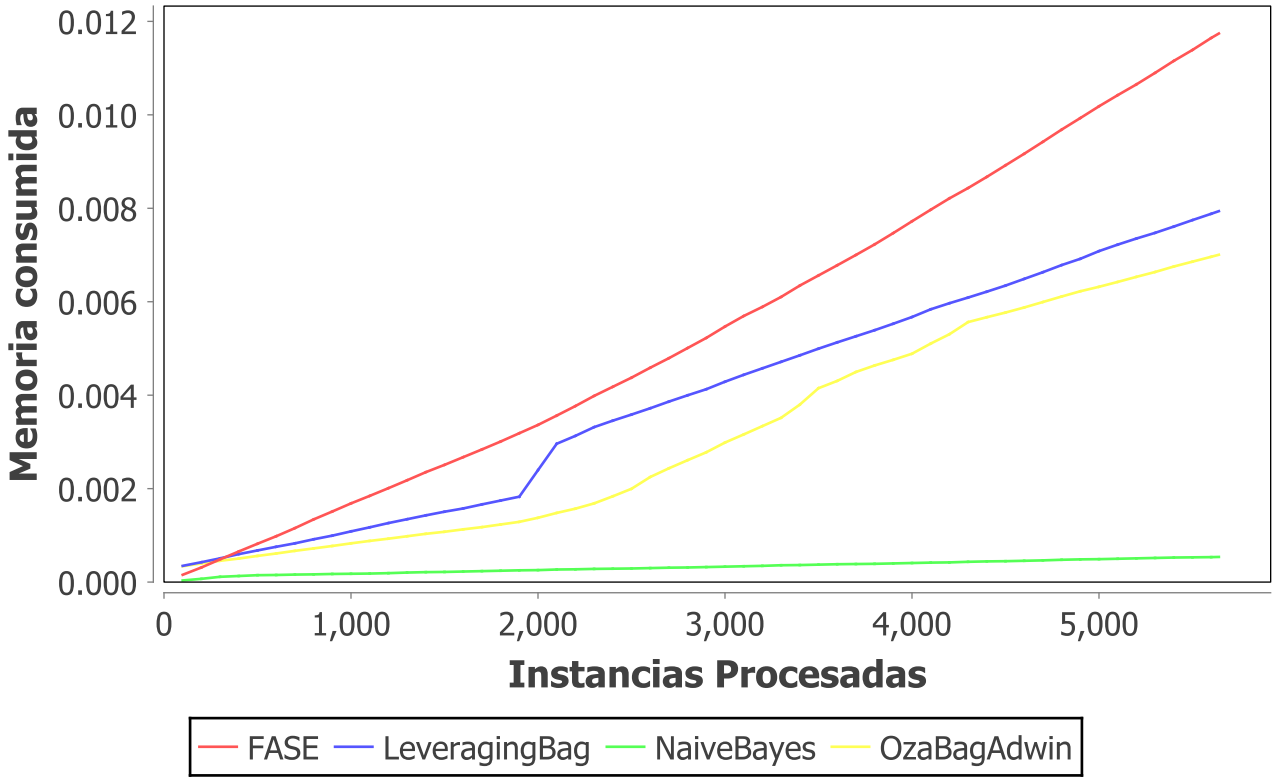


Figura B.1: Consumo de memoria de los algoritmos para la colección DS1

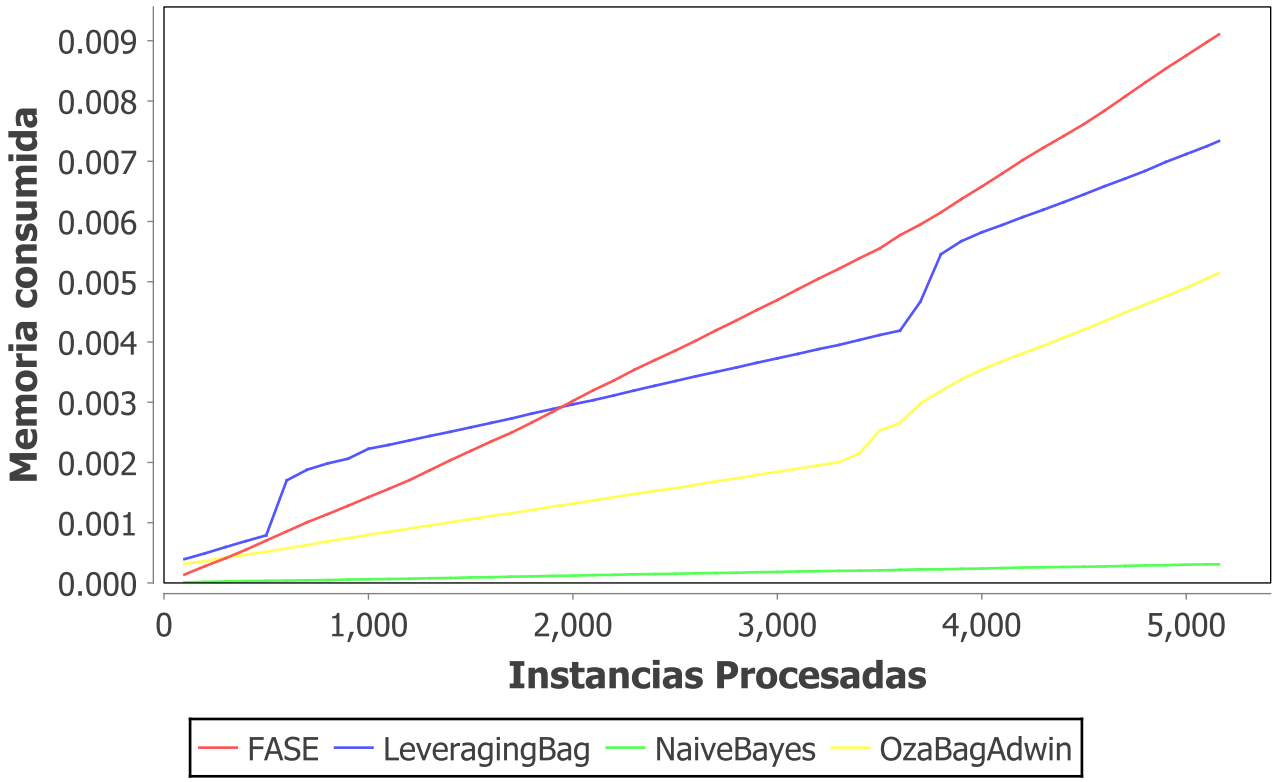


Figura B.2: Consumo de memoria de los algoritmos para la colección DS2

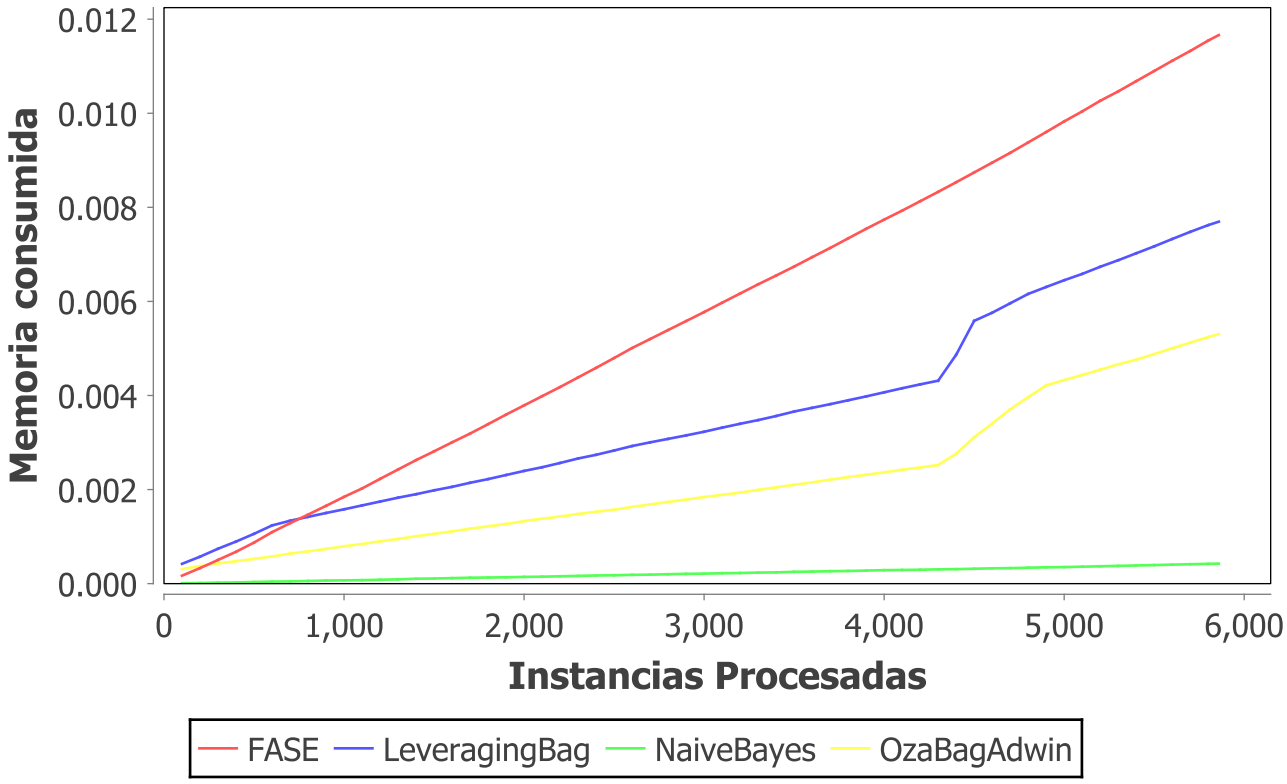


Figura B.3: Consumo de memoria de los algoritmos para la colección DS3

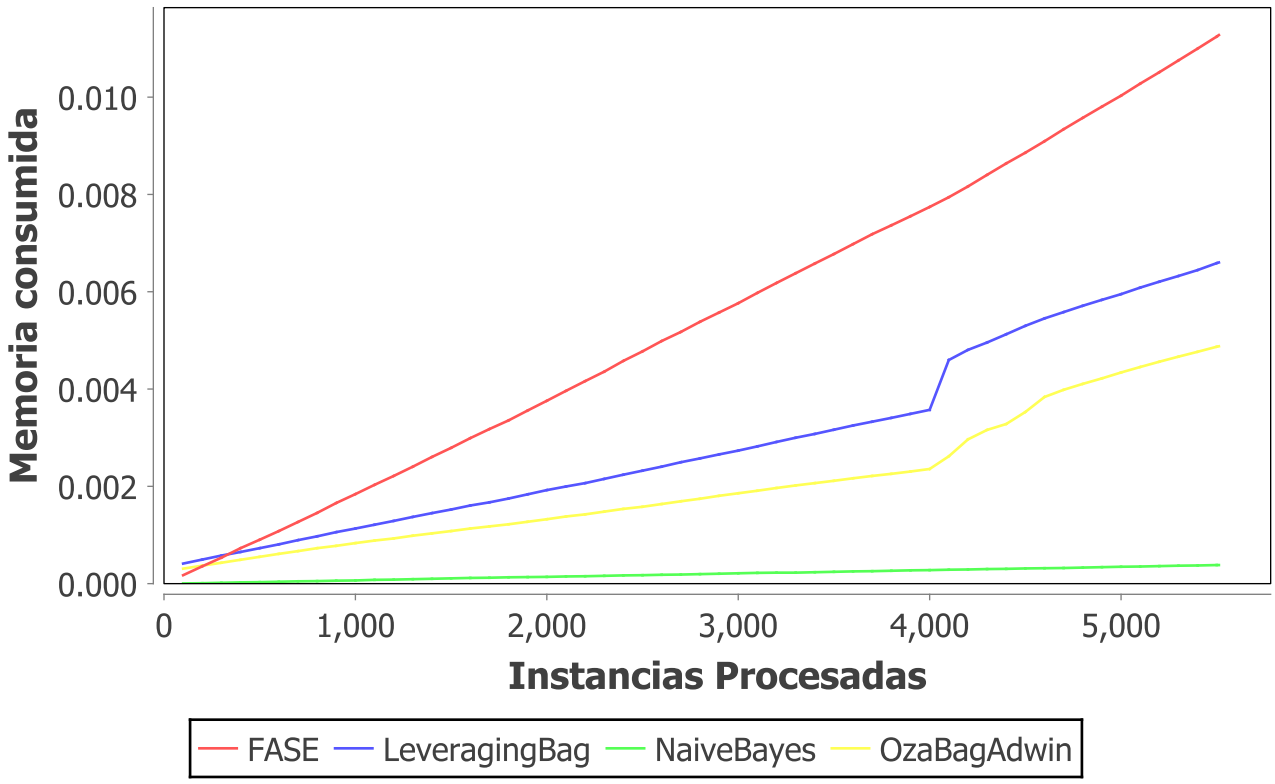


Figura B.4: Consumo de memoria de los algoritmos para la colección DS4

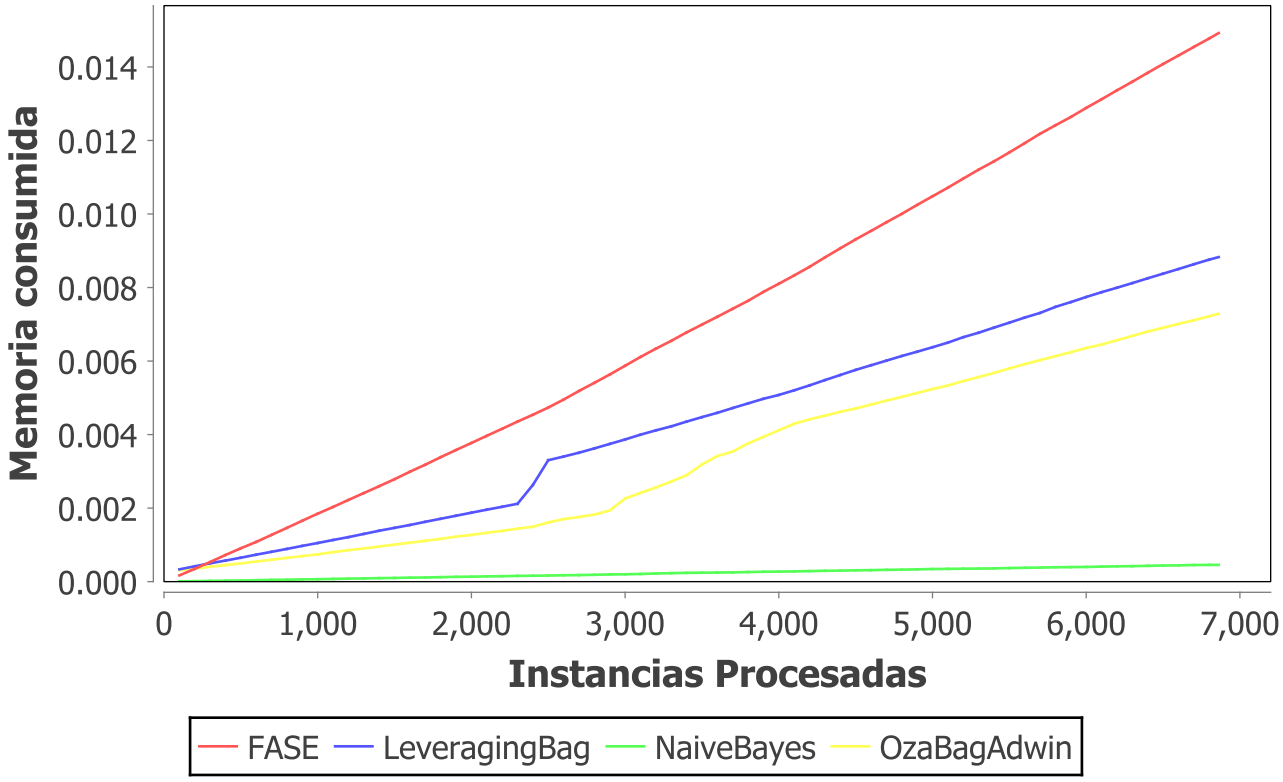


Figura B.5: Consumo de memoria de los algoritmos para la colección DS5

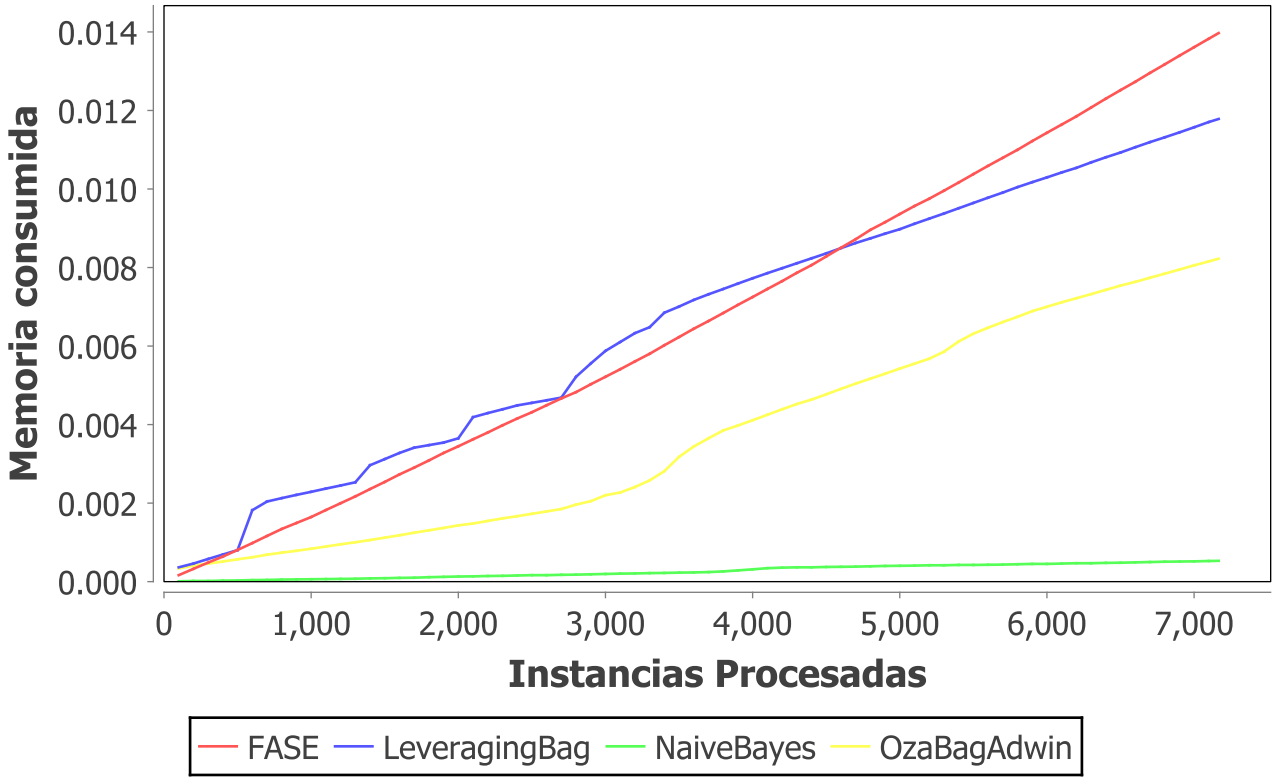


Figura B.6: Consumo de memoria de los algoritmos para la colección DS6

Consumo de memoria (en MegaBytes) de los algoritmos para una selección de 600 atributos

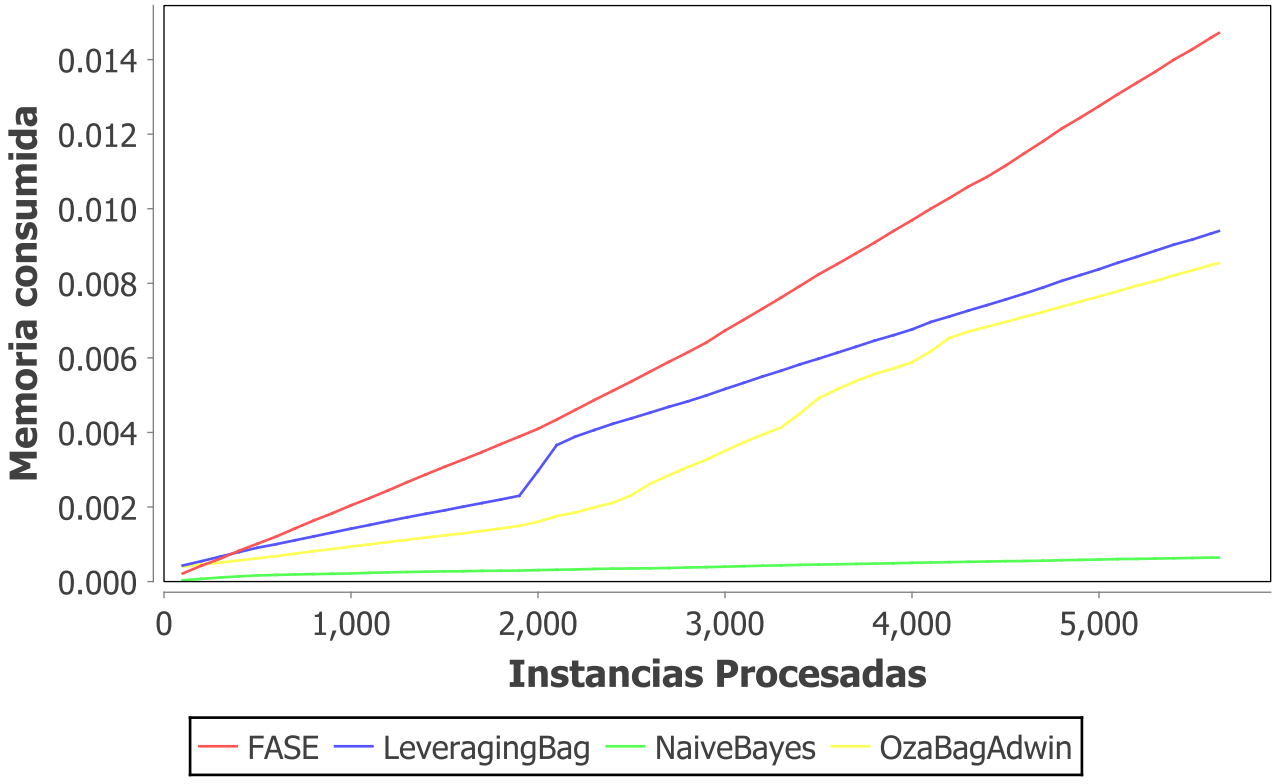


Figura B.7: Consumo de memoria de los algoritmos para la colección DS1

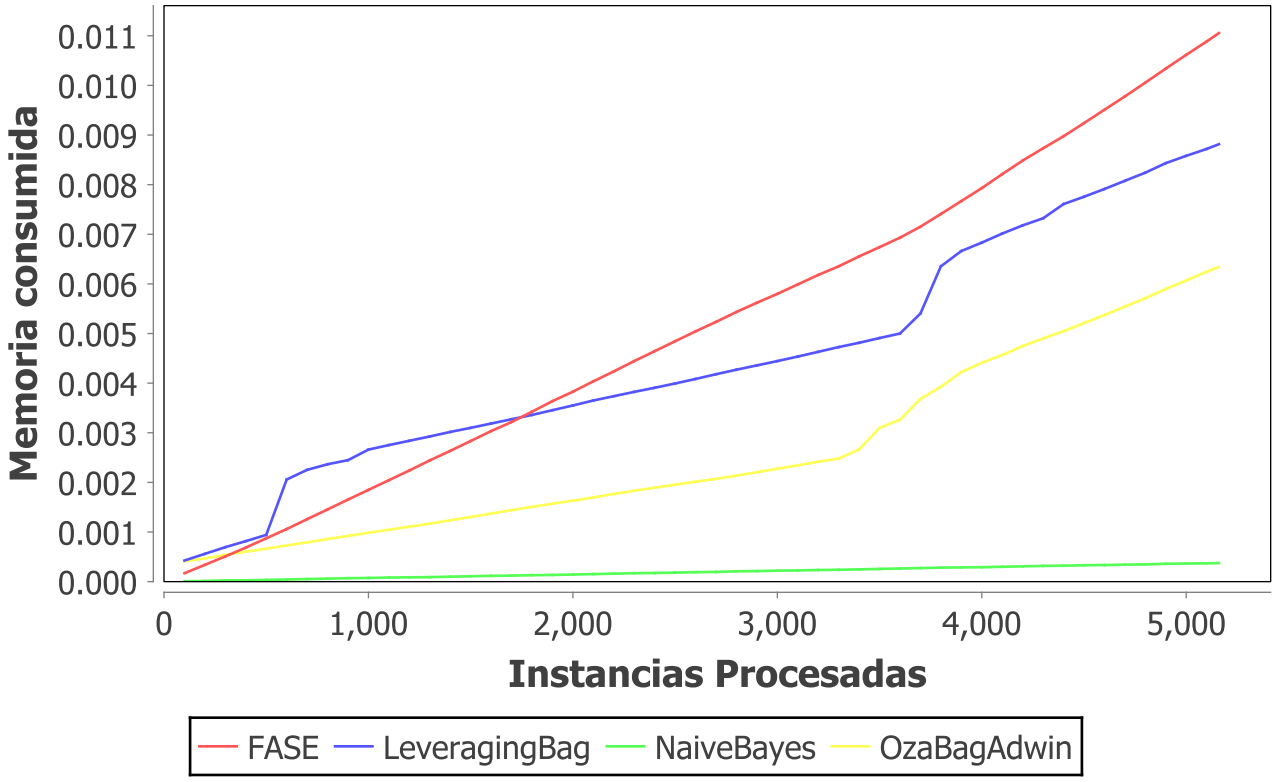


Figura B.8: Consumo de memoria de los algoritmos para la colección DS2

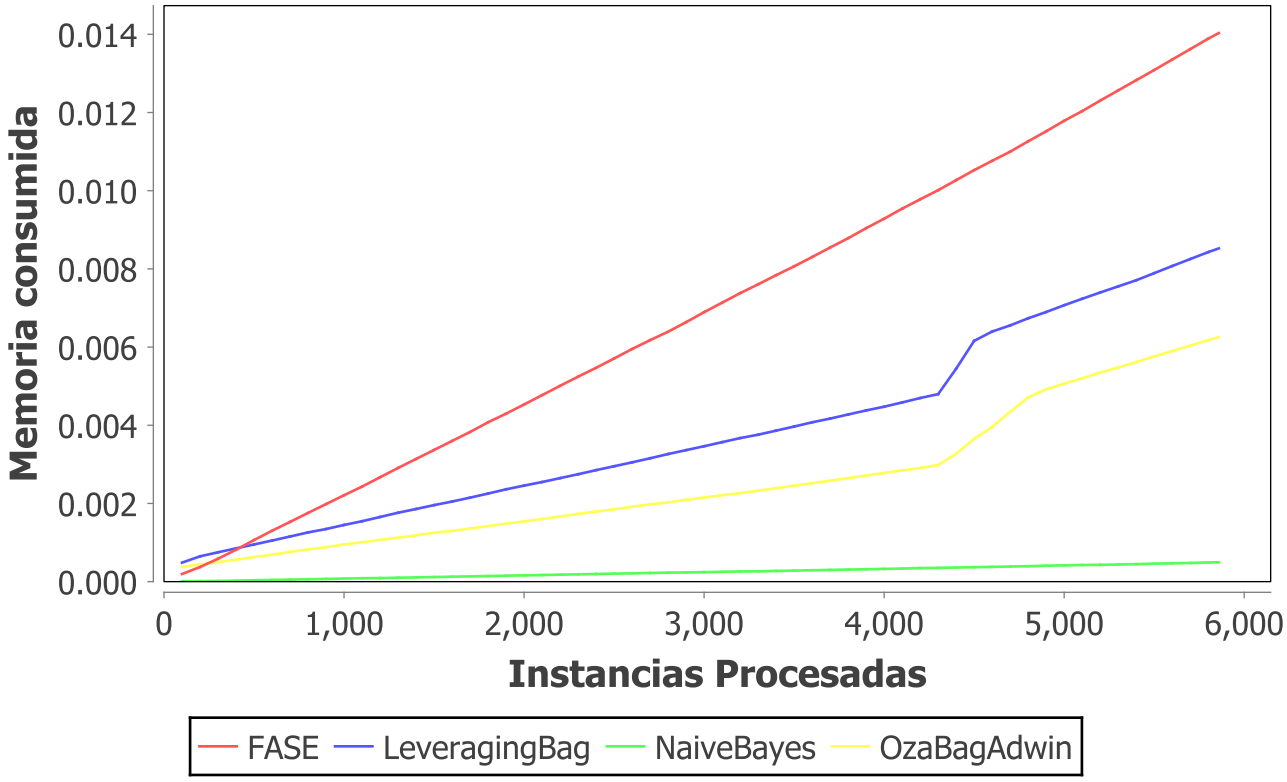


Figura B.9: Consumo de memoria de los algoritmos para la colección DS3

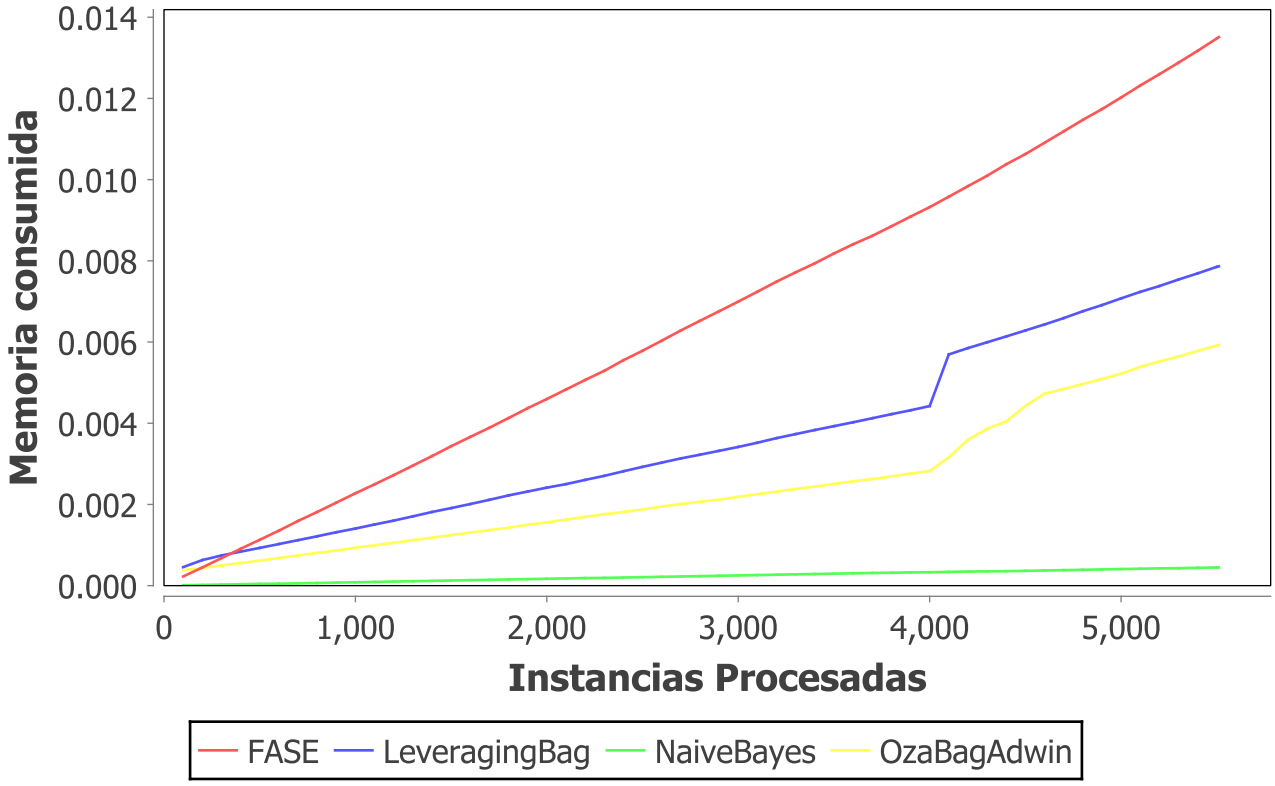


Figura B.10: Consumo de memoria de los algoritmos para la colección DS4

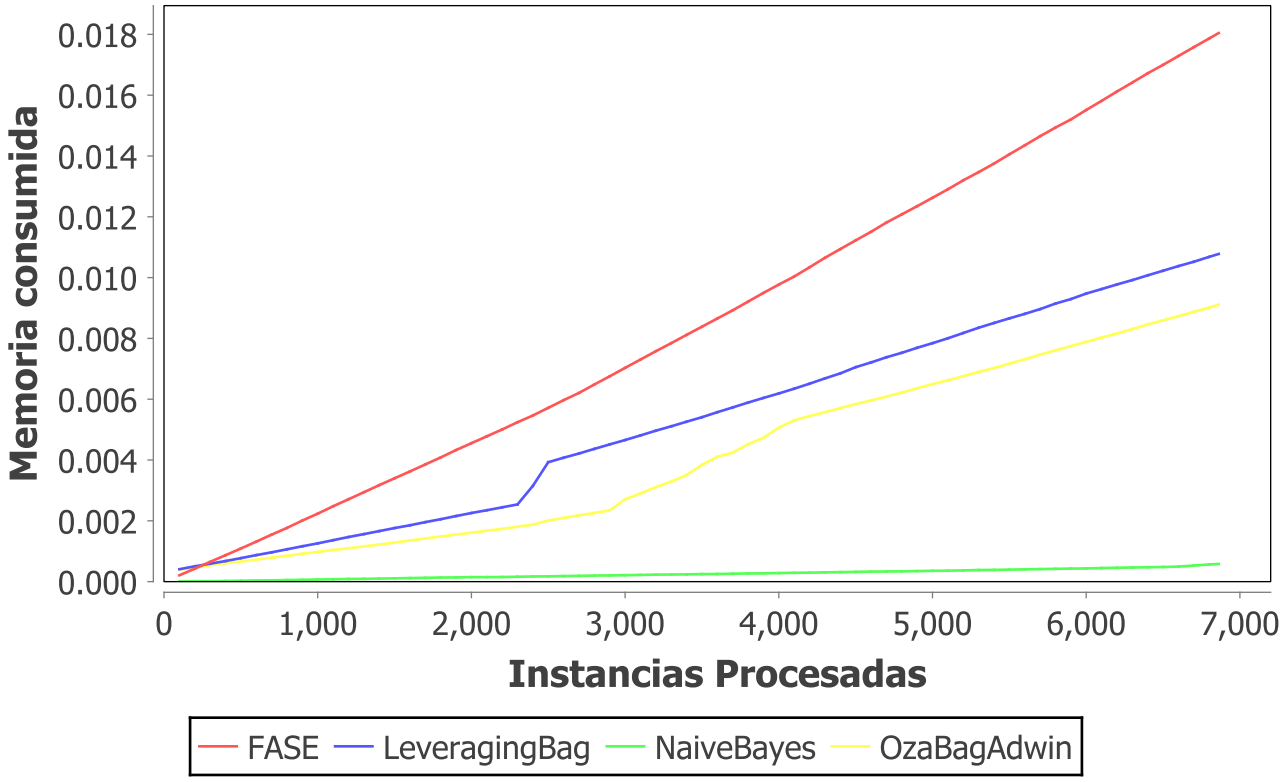


Figura B.11: Consumo de memoria de los algoritmos para la colección DS5

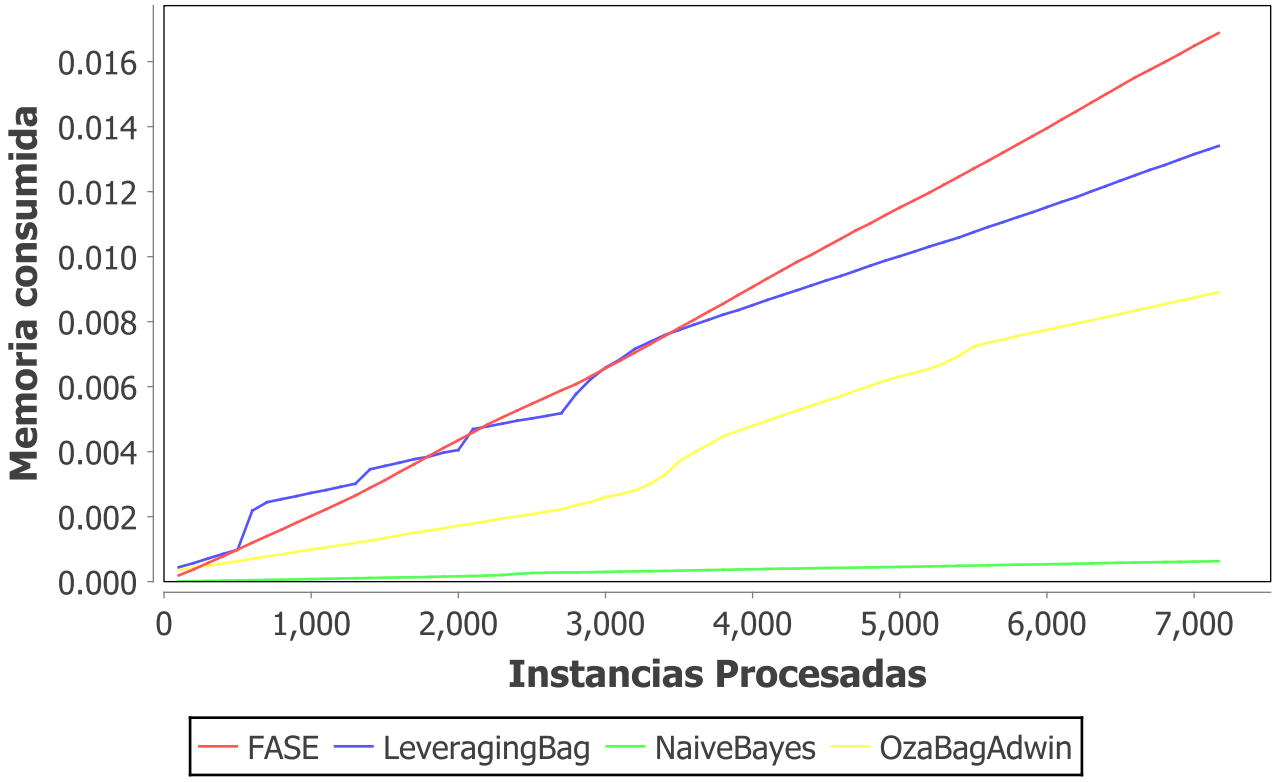


Figura B.12: Consumo de memoria de los algoritmos para la colección DS6