PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Informe Proyecto de título

Ingeniería Civil Informática

Análisis de sentimiento y clasificación de texto mediante Adaboost Concurrente

José Alfredo Poblete Castro
16.845.924-6

Año 2016

Prof. Guía: Rodrigo Alfaro
Prof. Co-Referente: Héctor Allende

Agradecimientos

Quiero expresar mi agradecimiento a las personas que directa e indirectamente han participado de mi desarrollo académico y de la presente tesis. En primer lugar a mi madre Olga Castro Jiménez, por su innegable y constante sacrificio para poder llegar a este punto de mi desarrollo personal y profesional. También agradecer a mis profesores guías Rodrigo Alfaro y Héctor Allende, por su colaboración, paciencia y apoyo durante todo el transcurso de esta tesis.

Índice

Lista de Figuras

1 Introducción

El siguiente informe pretende dar a conocer las experiencias obtenidas en la realización e implementación de la solución propuesta de Proyecto de Título, de la carrera de Ingeniería Civil Informática en la Pontificia Universidad Católica de Valparaíso, tomando como tema el de **"Análisis de sentimiento y clasificación de texto mediante Adaboost Concurrente".**

En el contexto actual, Internet se ha convertido en una fuente masiva y continua de opiniones. Estas opiniones suelen estar contenidas en textos escritos en lenguaje natural, ya sea en los reviews de productos escritos por clientes o por analistas profesionales, en los foros públicos existentes de diversas temáticas, en los blogs o más recientemente en determinados servicios de microblogging y redes sociales. El acceso y la explotación de estas nuevas fuentes de opinión poseen un indudable atractivo para las administraciones, las empresas y los clientes.

En los últimos años, diversos investigadores del campo del Procesamiento del Lenguaje Natural se han centrado en estudiar el tratamiento computacional de las opiniones, los sentimientos y otros fenómenos subjetivos contenidos en este tipo de textos. Esta nueva disciplina, conocida como Minería de Opiniones o Análisis del Sentimiento, es de gran utilidad en el contexto que acabamos de plantear. Muchas son las tareas que se han ido definiendo en estos últimos años: clasificación de documentos basada en la opinión contenida en los mismos, detección de la subjetividad las emociones expresadas en los textos, clasificación de documentos basada en la perspectiva política del autor, etc.

En este trabajo, nos centramos primeramente en la extracción de opiniones sobre la red social Twitter, aplicadas por ejemplo en las opiniones acerca de los candidatos en las elecciones presidenciales e industria del retail, y luego se procede a probar el algoritmo de adaboost concurrente, viendo su eficacia y resultados pre y post entrenamiento con datos y pruebas pre establecidas, para finalmente poder comparar y contraponer los casos estudiados.

2 Objetivos

2.1 Objetivo General

Como objetivo general de la propuesta, se tiene el de analizar, estudiar y comparar el desempeño del algoritmo de Adaboost Concurrente, en la clasificación de textos cortos de la red social de Twitter.

2.2 Objetivos Específicos

- Implementar, analizar y adaptar Adaboost concurrente.
- Utilizar métricas de desempeño como F1, Accuracy, Roc para determinar el éxito de la clasificación de textos cortos.

3 Problema a aplicar

3.1 Definición del problema

El fenómeno de las redes sociales pasó de ser un medio de interacción para jóvenes a ser un canal de comunicación que las empresas deben considerar seriamente. El estudio titulado “Social Media Benchmark Study 2013” nos dice que el 67% de los consumidores utilizan los perfiles en redes sociales de una empresa para obtener información sobre los servicios y que el 43% de los jóvenes de entre 18 y 29 años, prefiere interactuar con las marcas a través de las redes sociales, que utilizar otro tipo de comunicación. Es por esto que nace la problemática de cómo interpretar esta enorme cantidad de opiniones que los distintos clientes y usuarios de marcas y empresas están realizando en las redes sociales, denominando a este tipo de datos como una “divisa virtual” que gratuitamente proporcionan los usuarios de las redes sociales, y que actualmente se encuentra sin mucha explosión ni investigación. Así, se busca llegar a una forma de poder inferir si las opiniones efectuadas corresponden a una polaridad positiva, neutra o negativa, utilizando algortimos semi asistidos para esta tarea en particular. En definitiva, las redes sociales son un mercado en que las empresas deben invertir tiempo y esfuerzo.

Una gran parte de los contenidos generados por los usuarios en forma de textos escritos en lenguaje natural tiene un carácter subjetivo: los reviews de productos o servicios, las entradas de algunos blogs o los comentarios hechos por otros usuarios, los hilos de discusión en los foros públicos (acerca de política, cultura, economía o cualquier otro tema), los mensajes escritos en servicios de microblogging como Twitter o en redes sociales como Facebook. Todos estos son textos en los que los internautas pueden expresar sus opiniones, puntos de vista y estados de ánimo. Al igual que es frecuente apoyarse en las opiniones de nuestros conocidos a la hora de tomar determinadas decisiones (por ejemplo, que modelo comprar de un determinado producto, o que película ir a ver al cine), es cada vez más frecuente basar dichas decisiones en las opiniones que los internautas vuelcan en la red. Por otro lado, son evidentes las implicaciones prácticas de este tipo de contenidos para las compañías o administraciones públicas; por ejemplo, hoy día muchas compañías disponen de personal que se dedica a monitorizar las opiniones aparecidas en Internet acerca de los productos y servicios de la misma, de manera que puedan interferir activamente evitando la propagación de estas opiniones y sus posibles consecuencias negativas de cara a las ventas o a la imagen de la marca. Pero independientemente del objetivo buscado, el número de contenidos individuales que deberían ser considerados es de tal magnitud que se hacen imprescindibles ciertos mecanismos de procesamiento automático de los mismos. Si bien estamos de nuevo hablando del campo de actuación del PLN, el carácter subjetivo del tipo de contenidos que se pretende procesar presenta determinadas peculiaridades y da lugar a nuevos retos, poco estudiados tradicionalmente por el PLN. Se abre así una nueva subdisciplina conocida como Análisis del Sentimiento o Minería de Opiniones que viene a ocuparse del procesamiento computacional de este tipo de contenidos subjetivos.

Así, el análisis del sentimiento (sentiment analysis) o minería de opiniones (opinion mining) abarca aquellas tareas relacionadas con el tratamiento computacional de las opiniones, los sentimientos y otros fenómenos subjetivos del lenguaje natural. Algunas de estas tareas son la clasificación de documentos de opinión según el carácter positivo o negativo de las opiniones, la extracción de representaciones estructuradas de opiniones, o el resumen de textos de opinión, entre otras. Estas tareas tienen puntos en común con otras tareas clásicas del PLN, como pueden ser la clasificación de documentos, la extracción de información y el resumen automático. Sin embargo, el carácter subjetivo de los textos analizados añade ciertas peculiaridades a las tareas anteriores y las hace especialmente difíciles. Por ejemplo, la clasificación según Naive Bayes es diferente en su tratamiento y resultados a las SVM.

4 Técnicas utilizadas en aprendizaje automático

El Aprendizaje Automático, aka Machine Learning, es la rama de la Inteligencia Artificial que se dedica al estudio de los agentes/programas que aprenden o evolucionan basados en su experiencia, para realizar una tarea determinada cada vez mejor. El objetivo principal de todo proceso de aprendizaje es utilizar la evidencia conocida para poder crear una hipótesis y poder dar una respuesta a nuevas situaciones no conocidas.

Así, se presenta un tipo de aprendizaje semi supervisado llamado AdaBoost Concurrente. En primera instancia, se procede a explicar el algoritmo de AdaBoost creado por Freund y Schapire.

4.1 AdaBoost

AdaBoost, abreviatura de "Adaptive Impulso", es una máquina de aprendizaje meta-algoritmo formulado por Yoav Freund y Robert Schapire que ganó el prestigioso "Premio Gödel" en 2003 por su trabajo. Se puede utilizar en conjunción con muchos otros tipos de algoritmos de aprendizaje para mejorar su rendimiento. La salida de los otros algoritmos de aprendizaje de los clasificadores ('débiles') se combina en una suma ponderada que representa la salida final del clasificador impulsado. AdaBoost es adaptativa en el sentido de que los clasificadores posteriores débiles están ajustados a favor de esos casos mal clasificados por clasificadores anteriores. AdaBoost es sensible a los datos ruidosos y valores atípicos. En algunos problemas, sin embargo, puede ser menos susceptibles a los problemas que otros algoritmos de aprendizaje. Los estudiantes individuales pueden ser débiles, pero siempre y cuando el rendimiento de cada uno es ligeramente mejor que adivinar al azar (es decir, su tasa de error es menor que 0,5 para la clasificación binaria), el modelo final se puede probar a converger a una clasificador fuerte.

Cuando se introdujo en primer lugar, el *problema hipótesis impulsar* simplemente se refirió al proceso de convertir un aprendiz débil en un fuerte clasificador. "Informalmente, la hipótesis de impulsar pregunta si un algoritmo de aprendizaje eficaz que da salida a una hipótesis cuya actuación es sólo ligeramente mejor que el azar adivinar (es decir, un clasificador débil) implica la existencia de un algoritmo eficiente que da salida a una hipótesis de arbitraria exactitud, es decir, un fuerte clasificador."

Mientras que cada algoritmo de aprendizaje tiende a adaptarse a algunos tipos de problemas mejores que otros, y suele tener muchos parámetros y configuraciones diferentes para ajustarse antes de lograr un rendimiento óptimo en un conjunto de datos, AdaBoost (con árboles de decisión como los clasificadores débiles) se refiere a menudo como el mejor clasificador fuera de la caja. Cuando se utiliza con el aprendizaje árbol de decisiones, la información recogida en cada etapa del algoritmo AdaBoost acerca de la 'dureza' relativa de cada muestra de entrenamiento se

alimenta a la creciente algoritmo de árbol de tal manera que los árboles posteriores tienden a centrarse en más difícil de clasificar ejemplos. A diferencia de las redes neuronales y SVM, el proceso de formación AdaBoost selecciona sólo aquellas características conocidas para mejorar la capacidad de predicción del modelo, la reducción de dimensionalidad y, potencialmente, mejorar el tiempo de ejecución de funciones como irrelevantes que no necesitan ser calculadas.

La figura a continuación muestra un diagrama de flujo, el cual explica el funcionamiento de Adaboost.

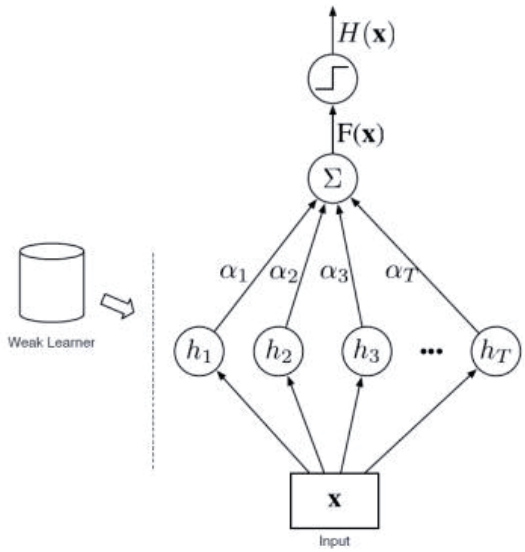


Figura 1, Diagrama de flujo Adaboost

A todos los datos se les asigna inicialmente el mismo peso de (1/m), este peso se va actualizando con cada iteración según los ejemplos mal clasificados, de esta manera se busca minimizar el error esperado y enfocarse en clasificar correctamente los datos que ahora tienen mayor peso. De este modo, al final se realiza la suma de todos los clasificadores previamente generados y se obtiene una hipótesis cuya predicción es más acertada.

La Figura 2 y 3 muestra un ejemplo del procedimiento que un algoritmo AdaBoost simple sigue para formar el clasificador final, para este caso se utiliza como weak learner un árbol de decisión de un solo nivel. Se tiene el siguiente conjunto de datos en donde un tipo de datos se representa por símbolos “+” azules y el otro por símbolos “-” en rojo. En su primer intento, el clasificador realiza un “corte” tratando de separar los datos pero podemos observar como realiza tres errores: deja dos datos negativos dentro del conjunto positivo y un dato positivo no lo toma en cuenta. En su siguiente intento, el peso de estos tres datos en donde hubo error es incrementado, de modo que el nuevo clasificador le de más importancia a resolver bien estos puntos. En su segundo intento logra discriminar correctamente a los datos negativos que tenían más peso pero ahora incluye aquellos que tenían un bajo peso. Debido a esto, se incrementa el peso de estos errores y,

por el contrario, aquellos datos que ya fueron correctamente clasificados reciben un peso menor. Se realiza un tercer clasificador que de igual manera tratará de hacer el mejor "corte" para resolver el problema. Finalmente se sumarán todos los clasificadores creados para crear la solución que, teóricamente, debería de ser mejor que cada clasificador por separado.

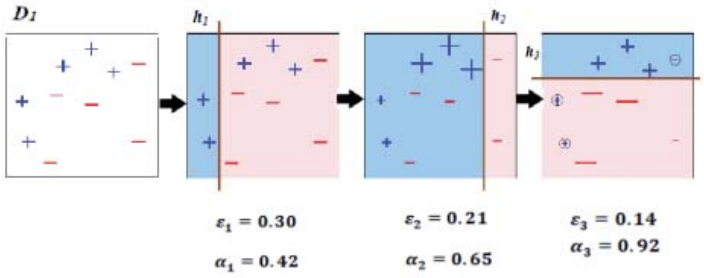


Figura 2, Ejemplo Adaboost simple

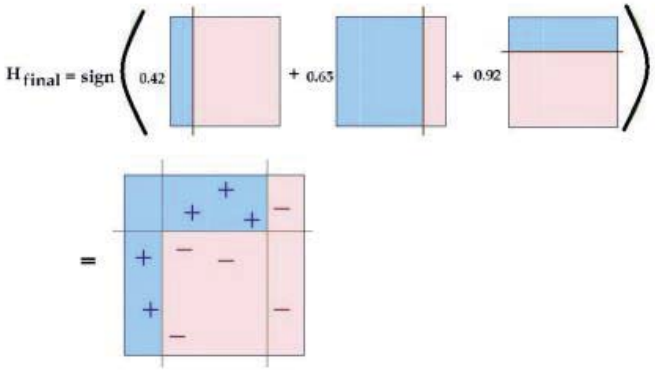


Figura 3, Ejemplo Adaboost simple

AdaBoost se refiere a un método particular de entrenamiento de un clasificador impulsado. Un clasificador Boost es un clasificador en la forma:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

Donde cada f_t es un aprendiz débil que toma un objeto x como entrada y devuelve un resultado de valor real que indica la clase del objeto. La señal de la salida del clasificador débil identifica la clase de objeto predicho y el valor absoluto da la confianza en esa clasificación. Del mismo modo, la T del clasificador de capa será positivo si se cree que la muestra a ser en la clase positivo y negativo de otra manera.

Cada clasificador débil produce una salida, la hipótesis $h(x_i)$, Para cada muestra en el conjunto de entrenamiento. En cada iteración t, se selecciona un aprendiz débil y le asigna un coeficiente α_t tal que el error de entrenamiento suma E_t de la resultante t del impulso clasificador se minimiza.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

Aquí $F_{t-1}(x)$ es el clasificador impulsado que se ha construido hasta la etapa anterior de la formación, $E(F)$ es alguna función de error y $f_t(x) = \alpha_t h(x)$ es la principiante débil que se está considerando para la adición al clasificador final.

En cada iteración del proceso de formación, se le asigna un peso a cada muestra en el conjunto de entrenamiento igual al error actual $E(F_{t-1}(x_i))$ en esa muestra. Estos pesos pueden ser utilizados para informar a la formación del clasificador débil, por ejemplo, los árboles de decisión pueden ser cultivados para los conjuntos de a favor dividir de las muestras con altos pesos.

A continuación se procede a explicarlo en forma más detallada, mostrando el algoritmo en cuestión con sus respectivos pasos:

Dado: $(x_1, y_1),\ldots, (x_m, y_m)$ donde $x_i \in$ X, $y_i \in \{-1,+1\}$.
Inicializar: D_1(i) = 1/m para i = 1,…,m.
Para t = 1,…,T:
• Entrenar al clasificador débil utilzando la distribución D_t .
• Recibir la hipótesis débil h_t : X → {−1,+1}.
• Objetivo: seleccionar h_t con bajo error ponderado: $\varepsilon_t = Pr_{iD_t}[h_t(x_i) \neq y_i]$.
• Elegir $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
• Actualizar, para i = 1,…,m:

$$D_{t+1} = \frac{D_{t(i)exp\left(-\alpha_t y_i h_t(x_i)\right)}}{Z_t}$$

Donde Z_t es el vector de normalización (elegido de modo que D_{t+1} será una distribución)
Como salida, finalmente se tiene la hipótesis:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

4.1 AdaBoost Concurrente

Se presenta a continuación la modificación de Adaboost llamada "Adaboost concurrente", publicada por el profesor de la PUCV Don Héctor Allende Cid, en colaboración de Carlos Valle de la Universidad UTFSM de Valaparaíso, Rodrigo Salas de la UV y Claudio Moraga del Centro Europero Soft Computing.

La mayoría de los ordenadores modernos hoy en día tienen procesadores con múltiples núcleos. Adaboost se propuso en un tiempo fueron el número de núcleos por máquina era más limitado. Por lo tanto, parece natural a utilizar todos los recursos disponibles para mejorar la calidad de la inferencia hecha por este modelo. Para esto se mejora la fase de estimación de peso, que es una de las etapas más importantes en el algoritmo AdaBoost, utilizando la computación concurrente basada en procesos paralelos.

La consigna principal (a diferencia de Adaboost simple) es trabajar con todos los procesadores de la máquina con el fin de mejorar la capacidad de generalización. En la mayoría de los enfoques AdaBoost paralelos, los múltiples núcleos se utilizan para disminuir los tiempos de cálculo, dividiendo el conjunto de datos en las fracciones más pequeñas. Estos enfoques tratan de obtener una aproximación al modelo que se habría obtenido se había entrenado con enfoques clásicos.

En esta propuesta, en lugar de utilizar un solo clasificador débil en cada ronda Adaboost, se utilizan todos los procesadores disponibles p en volver a muestrear, de una manera paralela, los datos originales, y también en paralelo varios clasificadores débiles. Con todos los p clasificadores débiles que construimos un conjunto, que se pondera con su exactitud entrenamiento, y luego con la salida del conjunto de actualizar las ponderaciones de los ejemplos.

En la propuesta, elegimos cada h^j según la Ec. de la figura 6, donde j = 1,.., P, y luego obtener la salida conjunto $E_t(x_i)$ para ejemplos x_i en la ronda t-ésima como:

$$E_t(x_i) = \text{sign}\left(y_i \sum_{j=1}^{p} \varphi^j h_t^j\,(x_i)\right)$$

Donde φ^j es la precisión de la formación débil hipótesis h_t^j y $\sum_{j=1}^{p} \varphi^j h_t^j\,(x_i)$ es la decisión de los p clasificadores débiles que fueron entrenados en paralelo. A continuación, el error ponderado se calcula teniendo en cuenta la salida del conjunto E_t usando $\varepsilon_t = P_{ri\sim D_t}[E_t(x_i) \neq y_i]$. Con esto, evitamos la necesidad de tener que seleccionar un clasificador de manera explícita y utilizar un conjunto de clasificadores en su lugar.

Para tener más claridad respecto a lo expuesto, se procede a explicar en los siguientes pasos el Algoritmo Adaboost Concurrente:

Algoritmo:

1. Teniendo en cuenta que es el conjunto de datos de entrenamiento Z = $\{(x_1 , y_1),..., (x_n , y_n)\}$ con n elementos, donde $x_i \in$ X y $y_i \in$ Y = {-1, 1}.
2. Inicialice los parámetros. Escoja el parámetro p (número de procesos paralelos) el número de rondas T y t = 0.
3. Iniciar la distribución empírica D_1(i) = $\frac{1}{n}$ para cada muestra de datos (x_i, y_i), i = 1...n.
4. Repita.
5. Incrementar t por uno.
6. Tomar p muestras de arranque Z_t^j de Z con la distribución D_t, donde j es el número de procesos en paralelo, con j = 1,..., p. (En paralelo).
7. Entrenar p clasificadores débiles h_t^j : X → {-1, 1} con las muestras de arranque Z_t^j como el set de entrenamiento. (En paralelo)
8. Generar un conjunto E_t con todos los clasificadores débiles h_t^j.
9. Calcule el error ponderado ε_t del conjunto de hipótesis débiles E_t como:

$$\varepsilon_t = P_{ri\sim D_t}[E_t(x_i) \neq y_i]$$

10. Calcule la importancia empírica del conjunto t-ésimo con la ecuación de α_t:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

11. Actualización de la distribución empírica como:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \text{ x } e^{(-\alpha_t \beta(i)\, y_i\, E_t(x_i))}$$

Donde Z_t es el factor de normalización.

12. La hipótesis fuerte H_t(x) en la etapa t está dada por:

$$H_t(\text{x}) = \text{sign}\ (\textstyle\sum_{t=1}^{t} \alpha_t E_t\ (x))$$

13. Clasificar el conjunto de datos de entrenamiento Z = $\{(x_1 , y_1),..., (x_n , y_n)\}$ con la hipótesis fuerte H_t(x), **hasta** que se cumpla el criterio de parada del algoritmo.
14. Salida: La hipótesis fuerte H_t(x).

En la siguiente figura, se puede apreciar la diferencia mayor entre el enfoque clásico y el concurrente. En el clásico se observa que en cada ronda un solo clasificador débil se entrena con un nuevo muestreo de los datos originales, y las actualizaciones de los pesos de las distribuciones se cambian utilizando las salidas del único clasificador débil. En el caso del enfoque concurrente, en cada ronda de boosting, las p remuestras obtenidas en una moda concurrente se utilizan para entrenar los p clasificadores débiles para construir un conjunto ponderado por la precisión del entrenamiento. Este conjunto se utiliza luego para actualizar los pesos de la distribución.

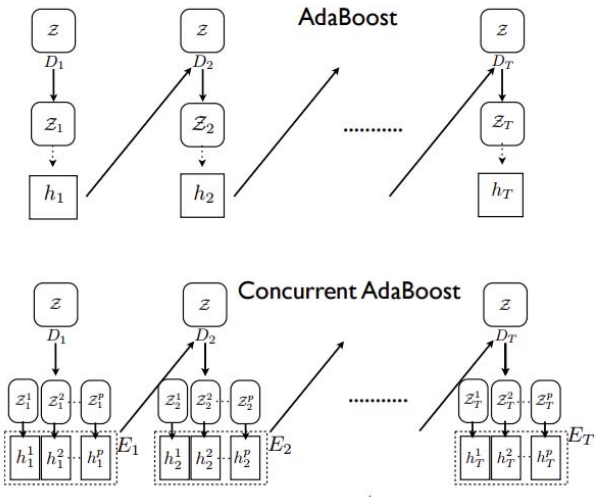


Figura 4, Enfoque AdaBoost Clásico y AdaBoost Concurrente.

Finalmente, Adaboost concurrente irá clasificando los mensajes cortos mediante un árbol de decisión, el cuale s un método de clasificación en el que el entrenamiento consiste en la construcción de un árbol de decisión de múltiples caminos en el que para cada nodo se busca el atributo que provee mayor ganancia de información para la clase. El árbol crece hasta su tamaño máximo y luego es acotado para mejorar su capacidad de generalización para los datos que no ocurren en el conjunto de datos de entrenamiento. A partir de este modelo se infieren reglas de decisión sobre las cuales se basa la clasificación.

5 Solución propuesta

Ante la problemática expuesta a lo largo de este informe, se decide separar en distintos pasos la solución propuesta para que la comprensión del lector sea óptima, por lo que se presentan las siguientes figuras con los mismos, seguidos por la explicación de cada uno de ellos:

5.1 Sentimiento

En términos generales, el análisis de sentimiento intenta determinar la actitud de un interlocutor o un escritor con respecto a algún tema o la polaridad contextual general de un documento. La actitud puede ser su juicio o evaluación, estado afectivo (o sea, el estado emocional del autor al momento de escribir), o la intención comunicativa emocional (o sea, el efecto emocional que el autor intenta causar en el lector).

Dentro de los tweets previamente clasificados en el set de datos, se muestra a continuación en el siguiente gráfico el sentimiento de las opiniones expresadas en twitter de la candidata Bachelet, en el transcurso del mes previo a las elecciones presidenciales del año 2013, entre las fechas de 17-10-2013 al 17-11-2013.

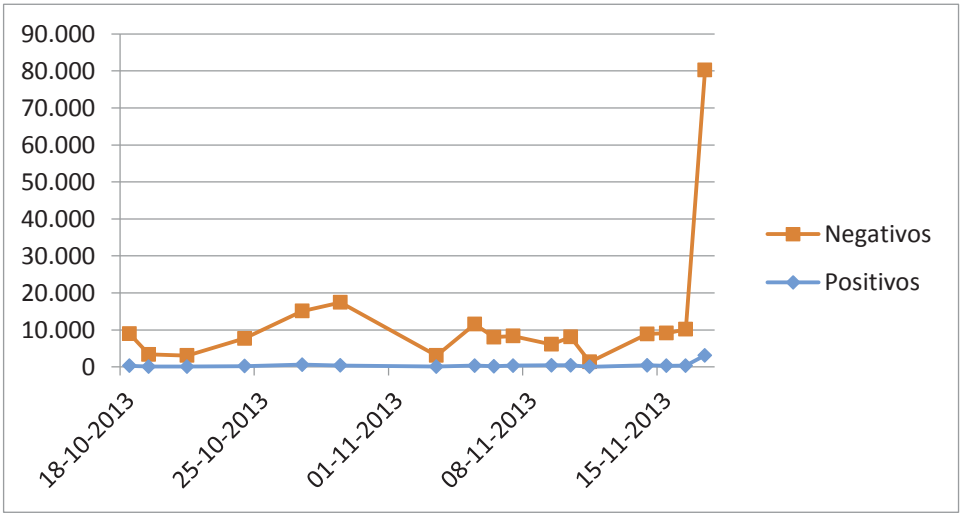


Figura 5, Sentimiento opiniones en twitter

5.2 Recolección de opiniones

Figura 6, Rescatar opiniones en Twitter

El proceso comienza con la extracción de opiniones de la red social de Twitter por la Empresa de la 5ta Región y conocida a nivel nacional "Analitic", la cual está dedicada al desarrollo de herramientas de monitoreo de Redes Sociales, y que en poco tiempo ha logrado posicionarse en el complejo mundo Social Media y Marketing Digital.

Los datos son obtenidos del período pre elecciones del año 2013, correspondientes a los meses de Octubre y Noviembre del mismo año. Para esto, Analitic utiliza sus distintos mecanismos para la extracción y creación del set de datos, los cuales se utilizarán en este proyecto.

Cabe señalar, que en el set de datos proporcionado por Analitic, ya existen un número de opiniones ya calificadas (como positivas, neutras o negativas), lo que servirá en las pruebas de entrenamiento de los algoritmos explicados en los pasos siguientes.

5.3 Carga de Tweets

Figura 7, Proceso de Carga de Tweets

El proceso continúa con la carga del set de datos en una base de datos MySQL que se ocupará a lo largo del proyecto. Para esto, se utiliza la herramienta "Pentaho Data Integration", la cual se define a sí misma como una plataforma de BI "orientada a la solución" y "centrada en procesos" que incluye todos los principales componentes requeridos para implementar soluciones

basados en procesos y ha sido concebido desde el principio para estar basada en procesos. Las soluciones que Pentaho pretende ofrecer se componen fundamentalmente de una infraestructura de herramientas de análisis e informes integrados con un motor de workflow de procesos de negocio. Para esto, se usa Spoon, que es una Interfaz Gráfica de Usuario (GUI) de Pentaho, que permite diseñar transformaciones y trabajos que se pueden ejecutar con las herramientas de la misma.

Finalmente, los datos se almacenan una base de datos MySQL, la cual se define como un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Dentro de la base de datos del proyecto, se crean set de datos (tablas) por separados de los candidatos presidenciales, que contienen los tweets en específico de cada uno de ellos, guardando distintas métricas de las opiniones, tales como usuario, fecha y hora de la opinión, etc.

5.4 Pre procesamiento y aplicación de Adaboost

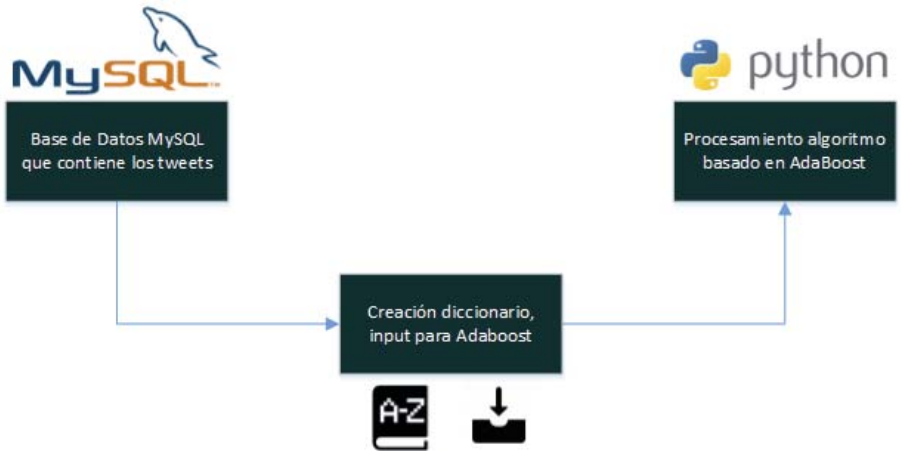


Figura 8, Pre procesamiento y aplicación Adaboost

Con la base de datos ya cargada, se procede en primera instancia a la creación del diccionario con todas las palabras existentes en los tweets, previa limpieza de palabras blancas (stopwords) y/o caracteres que no se deseen ingresar (explicados en el punto 6 de este informe), esto se procesa en lenguaje python (igual que el paso que sigue en esta parte de pre procesamiento). También, se crea el input en formato txt de manera matricial, asignando en esta los pesos TF-IDF de la representación del tweet, y su ponderación correspondiente.

Finalmente, luego de la creación del input, se procede a llamar a la clase de adaboost almacenada en el archivo “boosting.py”, en el cual se procederá a procesar y ejecutar el algoritmo que se ha expuesto a lo largo de este informe, arrojando como resultado distintas métricas y resultados, que se expondrán con detalle en el punto de aplicación de la solución, mostrando códigos y consideraciones para todo el proceso expuesto en el punto 5.

6 Aplicación de la Solución Propuesta

Para aplicar la solución que se ha expuesto en el punto 5, se procederá a desglosar y explicar de una manera más técnica en este apartado, teniendo como referencia el flujo de los distintos pasos y procedimientos escritos en lenguaje Python, screenshots que permitan observar tablas o representaciones de los datos y/o diagramas que surgen a partir de la implementación, permitiendo al lector tener una apreciación más gráfica.

6.1 Creación Set de Datos

Se tiene como base la información proporcionada por la Empresa Analitic, la cual consiste en archivos separados por candidatos (9) sumando un total de 2.000.000 de tweets aproximadamente, guardadas en tablas HTML del tipo *<table><tr>DATO 1</tr><tr>DATO 2</tr><tr>DATO n</tr></table>,* debiendo en primera instancia tener que transformarlas a formato .xlsx para su carga a una base de datos.

La principal problemática de este punto ocurre en los candidatos que poseen gran cantidad de tweets (Parisi, Bachelet, Matthei, Claude) conlleva a que el archivo no pueda abrirse y transformarse a formato Excel, debido a su gran tamaño, provocando que la carga del mismo termine por colapsar. Para solucionar esta problemática, y luego de intentar variadas formas, se decide por separar (cortar – pegar) manualmente el archivo desde su código fuente usando un editor de texto (Sublime text), dejando de esta forma archivos HTML más livianos. Posterior a esto, la carga del archivo HTML es exitosa, pudiendo así convertirlo a .xslx y dejando como outputs archivos de candidatos separados por partes.

Cabe destacar, que para probar el algoritmo en cuestión, se decide por crear un archivo de la candidata Bachelet, con 10.000 tweets positivos y negativos.

6.2 Carga en localhost de tweets

Para realizar esta tarea se utiliza la herramienta Pentaho Data Integration, pudiendo conectar el .xlsx con los datos de los candidatos con la base de datos MySQL alojada en un ambiente local (localhost), antes de su migración al servidor en el cual correrá la técnica semi supervisada de AdaBoost.

El flujo de carga, básicamente consiste en la transformación y asignación de variables por parte de Pentaho de las entradas (por partes) de los archivos .xlsx hacia la base de datos, por ejemplo, el (los) excel (s) que contengan los tweets de Bachelet se irán traspasando solo a la tabla

Bachelet en la base de datos del proyecto. A continuación se presenta la transformación .ktr de Pentaho de carga de tweets.

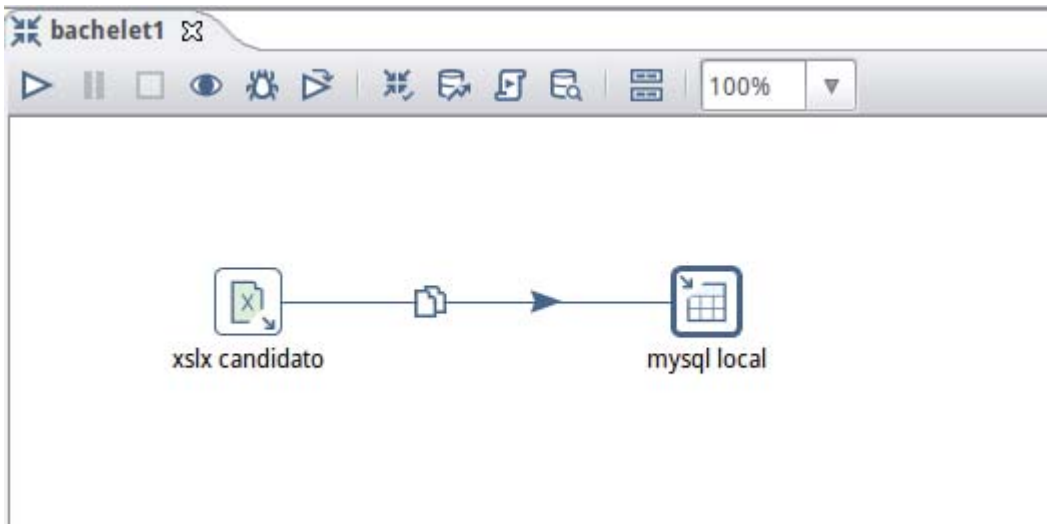


Figura 9, Carga de archivos a través de Pentaho

En la siguiente figura se puede observar el total de tweets cargados en la base de datos localhost a través de la herramienta de PhpMyAmin.

Figura 10, Base de datos MySQL en localhost

6.3 Stopwords, carácteres y excepciones de palabra

Para el correcto uso del algoritmo de Adaboost, se requiere limpiar de cierta medida las palabras usadas en los tweets por los usuarios, para la posterior creación de un diccionario con todas estas (explicado en punto 6.4). Para solucionar esta problemática, se crean funciones en Python las cuales permitirán limpiar y evaluar las palabras, y se explican de manera más profunda a continuación en los siguientes párrafos.

Las stopwords es el nombre que reciben las palabras sin significado como artículos, pronombres, preposiciones, etc. que son filtradas antes o después del procesamiento de datos en lenguaje natural (texto). Para este proyecto, se dispone de un conjunto de stopwords de 1136 palabras, las cuales son el resultado de una búsqueda y unión de varios diccionarios de este tipo de palabras existentes en la red. Se tienen por ejemplo dentro de este conjunto: *algún, alguna, algunas, alguno, algunos, ambos, ampleamos, ante, antes,* etc. Para la carga de estas, se tiene la función cargarStopWords(), la cual carga el txt de stopwords y retorna un array con este cargado, la cual será instanciada cada vez que el programa lo requierese para limpiar las palabras.

La función en Python "*excepcionPalabra(palabra)*" tiene como objetivo buscar y evitar insertar palabras que no queremos que se inserten en nuestro diccionario, recibiendo como parámetro cada una de las palabras existentes en los tweets cargados. En primera instancia, la función ocupa la subfunción de la clase string "*find*", el cual retorna un valor mayor a 0 en caso de que se encuentre lo que estamos buscando como parámetro, por ejemplo:

```
if palabra.find("@")>=0 or palabra.find("http")>=0 or palabra.isdigit()
```

En este caso, la función retornará *TRUE* en caso de que la palabra que ingresa contenga valores como "@", "*http*" o en caso de que sea un dígito (0, 1.., 9), evitando así el ingreso de usuarios de twitter, páginas web o números al diccionario. Adicionalmente, la función retornará *TRUE* en caso de que encuentre un dígito en el string, por ejemplo: "*casa9*"; en tal caso tampoco se ingresará al diccionario de palabras.

Finalmente, se tiene la función "*eliminarCaracter(palabra)*", la cual elimina los caracteres que acompañen a la palabra, por ejemplo:

```
if palabra.find('!') >=0:
        palabra=palabra.replace('!' ,"")  #elimina caracter " ! "
```

En este caso, a través de la sub función de la clase string "*find*" se busca si existe el carácter "*!*" que en caso de encontrarse se eliminará del string. Así, en caso de que entre un string del tipo "*viva!*", la función retornará solo el string "*viva*".

Cabe consignar que en caso de que la palabra fuese requiera un tipo de operación, está se retornará a la función de creación del diccionario, la cual se detalla en el punto a continuación.

6.4 Creación diccionario de palabras

Luego de completar los puntos anteriores, se procede a desarrollar el diccionario que contendrá el set de las palabras existentes en los 10.000 tweets positivos y negativos escogidos al azar de la candidata Bachelet. Para entender un poco mejor, se adjunta a continuación el código en lenguaje Python de esta función:

```
def crearDiccionarioTxt():        #crea (o agrega) diccionario de palabras respecto a tweets de candidato
   archivo = codecs.open("diccionario2.txt", "a+", "utf-8")
   stopwords = cargarStopWords()
   diccionario=[]
   candidatos =["bachelet"]

   for candidato in candidatos:
      tweets=rescatarTweetsCandidato(candidato)


      for tweet in tweets:      #se recorren los tweets
         temp=tweet[0]       #se rescata el cuerpo del tweet en la variable temp
         try:
            palabras=temp.split(" ")   #separa las palabras y las guarda en un array palabras

            for palabra in palabras:      #se recorre por palabra
               palabra=palabra.lower()        #eliminacion de mayusculas
               palabra=eliminarCaracter(palabra)              #eliminacion de caracteres que acompañan a la palabra

               if palabra in diccionario  or excepcionPalabra(palabra) or palabra in stopwords:
                  print 'palabra >>'+palabra+'<< No Ingresada'
                  #raw_input("presione una tecla")
               else:
                  archivo.write(palabra+'\n')              #se escribe en diccionario.txt
                  diccionario.append(palabra)             #se agrega al diccionario
                  print 'palabra >>'+palabra+'<< ingresa al diccionario \n'
         except:
```

```
            print 'error \n'
    archivo.close()

    return True
```

Del código anterior, primero se crea el archivo "*diccionario.txt*" y el array el cual tendrá el diccionario llamado "*diccionario=[]*", que estará vacío en un comienzo. Luego, se va recorriendo y rescatando los tweets por candidato, y al recorrer por cada tweet este se separa por palabras a través de la sub función de la clase String llamda "s*plit*", se eliminan las mayúsculas y caracteres y se comprueba si esta palabra ya existe en el array diccionario[] (para evitar repetir la palabra), si corresponde a una excepción o es una stopword; en tal caso la palabra no se ingresará al diccionario. En caso contrario, luego de pasar todas las condiciones mencionadas, la palabra es apta para ingresar al diccionario, escribiéndola en el txt y añadiéndola al array de palabras en diccionario. Finalmente, se obtiene el txt de diccionario, el cual contiene 9954 palabras. Cabe destacar que para llegar a este número, se eliminan hashtags "#", risas mal escritas y largas como "jaskjajka" y/o palabras redundantes (aproximadamente 1500 palabras).

Para el mejor uso de este diccionario, se procede a ordenarlo alfabéticamente con la función "*ordenarDiccionario()*", la cual a través de la subfunción "s*ort*" de la clase String de python ordena automáticamente el diccionario de la siguiente forma:

```
    archivo = codecs.open("diccionarioOrdenado.txt", "a+", "utf-8")

    diccionario=crearMatrizDiccionario()
    diccionario.sort()

    for palabra in diccionario:
       archivo.write(palabra+'\n')

    return True
```

6.5 Representación TF-IDF

Es la más utilizada para clasificar textos. El valor TF-IDF aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia

de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras, en donde la primera sección TF corresponde al valor de la frecuencia del término normalizado, multiplicado por IDF, que corresponde a la frecuencia inversa del término en la colección completa N.

6.6 Colección de documentos para TF-IDF

Para crear la colección de documentos de TF-IDF, se utiliza la librería "*python-tf-idf*", la cual se deben ir asignando los documentos a la colección (para luego ir pidiendo las ponderaciones en punto 6.6). Para esto se instancia un objeto de esta clase, un contador que servirá de identificador en la colección y un array que contendrá el tweet "limpio" luego de comprobar que no es un stopword y que no contenga caracteres de la siguiente forma:

```
table = tfidf.tfidf()
contador = 1
tweetLimpio=[]
```

La comprobación para ir añadiendo las palabras al array de tweetLimpio[], se realiza de la siguiente manera en lenguaje Python:

```
if excepcionPalabra(palabra) or palabra in stopwords:
        print 'palabra >>'+palabra+'<< No Ingresada a array de tweet limpio'
else:
        tweetLimpio.append(palabra) #se agrega palaba a tweetLimpio
        print 'palabra >>'+palabra+'<< ingresa al array de tweet limpio \n'
```

Vale decir, en caso de que la palabra pase las excepciones, se agrega al array de tweetLimpio, con la subfunción "*append*". Finalmente, cuando ya se posee el tweet "limpio" se procede a hacer la inserción a la colección de la siguiente manera:

```
table.addDocument(contador, tweetLimpio) #se ingresa el tweet limpio a la lista
contador=contador+1
tweetLimpio=[] #se resetea la variable tweetLimpio para prox iteración
```

Donde table.addDocument recibe como parámetros el id del elemento de la colección (que irá iterando) y el array del tweet limpio. Finalmente se resetea la variable de tweetLimpio para que se vuelva a ocupar en la próxima iteración con el siguiente tweet.

6.7 Asignación pesos TF-IDF y creación matriz principal

Luego de que la colección de documentos está concluida, se procede a asignar los pesos de las distintas palabras que poseen los tweets, asignándose luego a la matriz principal, que en su otra dimensión poseerá el arreglo de palabras de diccionario confeccionado con anterioridad en el punto 6.4, y que servirá como entrada al algoritmo de Adaboost.

Como primer paso, se instancia el arreglo de palabras de diccionario con la función *crearMatrizDiccionario()*, la cual retornará el array de todas las palabras existentes en los tweets realizados con anterioridad en el punto 6.4. Una manera gráfica de poder representar esto (teóricamente) sería de la siguiente manera:

A	B	C	D	E	F
tweets/palabras	viva	candidato	uno	odio	dos
viva el candidato uno	peso	peso	peso	peso	peso
odio al postulante dos	peso	peso	peso	peso	peso

Figura 11, Representación teórica matriz principal

Como se aprecia de la figura, en el eje y contendrá el tweet (o identificador) correspondiente, y en el eje contrario la palabra representada con su respectivo peso asignado por TF-IDF. Para la asignación del peso, la librería "*python-tf-idf*" a través de su subfunción "*table.similarities*", va actualizando los respectivos pesos con valores que van entre el 0 y el 1.

En el punto 6.8, se le agregará una columna de "Sentido" la cual servirá para que algoritmo de Adaboost sepa la ponderación de dicho tweet y saber si está ya clasificado o no (positivo, negativo).

6.8 Carga de tweets en servidor Pucv

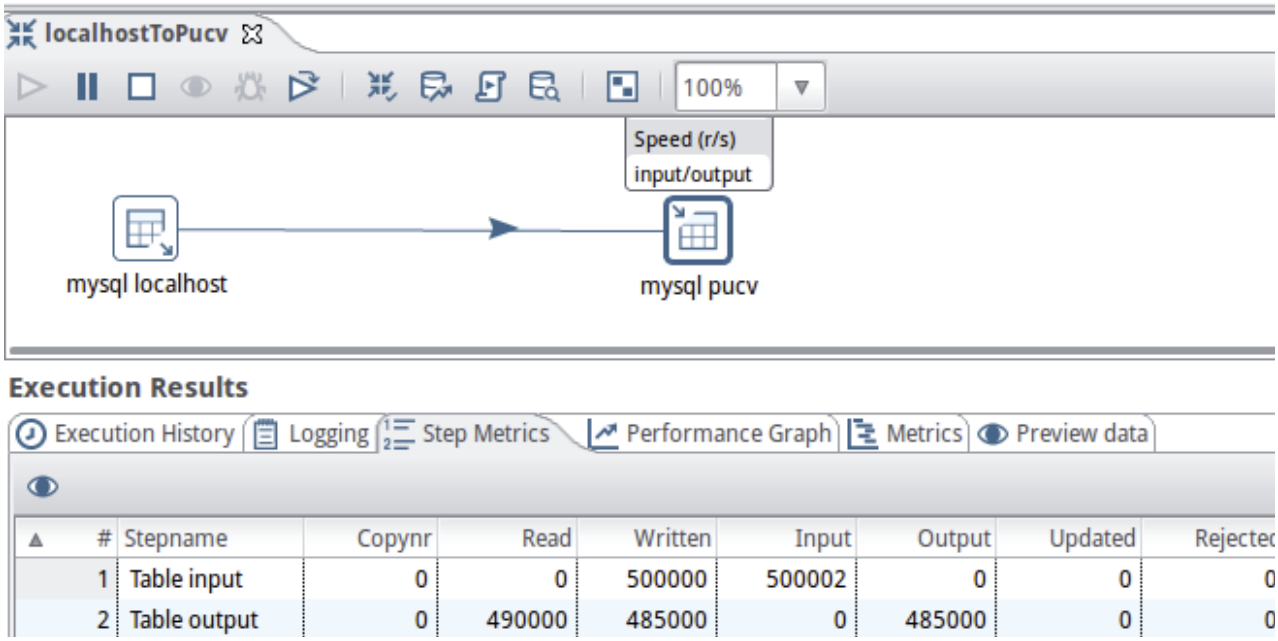
Para que algoritmo de Adaboost pueda ir actualizando las ponderaciones y rescatando los tweets de los candidatos, se debe migrar la base de datos alojada en localhost hacia el servidor alojado en la PUCV proporcionado por el profesor Héctor Allende-Cid. Para esto, se deben hacer 2

túneles SSH para la visualización de los datos (puerto 80), y la conexión de la base de datos (3306), esto se realiza de la siguiente manera (utilizando consola de Linux):

```
ssh -D 9000 jose@158.251.88.178          #túnel que servirá para recibir el puerto 80 pucv en
                                          el puerto 9000 de localhost

ssh jose@158.251.88.178 -L 3309:127.0.0.1:3306 -N -f proyecto     #túnel para base de
                                                                   datos en MySQL, puerto 3306
                                                                   en pucv, 3309 en localhost
```

Ya realizado estos pasos, se procede a migrar la base de datos alojada en localhost hacia el servidor Pucv mediante la herramienta Pentaho, de la siguiente manera:



#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected
1	Table input	0	0	500000	500002	0	0	0
2	Table output	0	490000	485000	0	485000	0	0

Figura 12, migración tweets desde localhost a servidor Pucv

6.9 Inputs AdaBoost

Para la carga en Adaboost de una manera de que el input sea correcto, se decide la opción de crear un txt para la que el archivo boosting.py pueda cargar el archivo y cargar la matriz en memoria. Se usará la siguiente forma (por cada tweet):

0, 0.2, 0. 0, 0, ,0 ,0 ,0 ,0, 0, 0, 0,…….,0 3343, 0, 0, 0.3, 0; -1

En tal caso, se observa en tal input que los primeros valores corresponden a valores entre 0 y 1, los cuales corresponderán al peso TF-IDF de la palabra correspondiente en el tweet. Para esto, primero se instancia el array de diccionario (que posee el diccionario de todas las palabras) con 0, para luego ir buscando la palabra correspondiente y asignandole el peso que otorga la clase de tf-idf de python. Luego, al final del input se observa un ";" que corresponderá al sentido del tweet, el cual será -1 para negativo y 1 para positivo.

6.10 Aplicación algoritmo Adaboost en servidor PUCV

Para la aplicación del algoritmo en el servidor PUCV, se procede a conectar vía SSH a través de Putty, realizando los distintos procedimientos en python para la creación de diccionario y txt de input para Adaboost. Luego de esto, se procede a llamar al archivo que contiene el algoritmo, proporcionándole por parámetro el numero de rondas (20) que se utilizarán.

Para la visualización de los resultados post aplicación del algoritmo, se proceden a explicar en el punto 7 del presente informe.

7 Resultados

Los experimentos son llevados a cabo en el servidor PUCV explicado en el punto 6. El servidor en cuestión, posee un procesador Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz, 16 Gigas de memoria Ram, Sistema operativo Ubuntu 14.04.3 LTS.

La data consiste en 10.000 tweets de la candidata Bachelet con ponderaciones positivas o negativas, se implementa en lenguaje Python 2.7.6 y se utiliza un 80% de los datos entrenar el modelo y un 20% para el testeo. Por cada porcentaje de distribución escogido, se realizan 6 experimentos con 20 rondas cada uno.

Las métricas con las que se trabaja y se analiza el desempeño del algoritmo son las sgtes:

i) Accuracy: Hace referencia a la conformidad de un valor medido con su valor verdadero, es decir, se refiere a cuán cerca del valor real se encuentra el valor medido. En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. Cuanto menor es el sesgo más exacto es una estimación. Cuando se expresa la exactitud de un resultado, se expresa mediante el error absoluto que es la diferencia entre el valor experimental y el valor verdadero.

ii) ROC (Area under the roc curve): El análisis ROC se relaciona de forma directa y natural con el análisis de coste/beneficio en toma de decisiones. Consideremos un problema de predicción de clases binario, en la que los resultados se etiquetan positivos (p) o negativos (n). Hay cuatro posibles resultados a partir de un clasificador binario como el propuesto. Si el resultado de una exploración es p y el valor dado es también p, entonces se conoce como un Verdadero Positivo (VP); sin embargo si el valor real es n entonces se conoce como un Falso Positivo (FP). De igual modo, tenemos un Verdadero Negativo (VN) cuando tanto la exploración como el valor dado son n, y un Falso Negativo (FN) cuando el resultado de la predicción es n pero el valor real es p. Un ejemplo aproximado de un problema real es el siguiente: consideremos una prueba diagnóstica que persiga determinar si una persona tiene una cierta enfermedad. Un falso positivo en este caso ocurre cuando la prueba predice que el resultado es positivo, cuando la persona no tiene realmente la enfermedad. Un falso negativo, por el contrario, ocurre cuando el resultado de la prueba es negativo, sugiriendo que no tiene la enfermedad cuando realmente sí la tiene.

   A modo de guía para interpretar las curvas ROC se han establecido los siguientes intervalos para los valores de AUC:

[0.5, 0.6): Test malo.
[0.6, 0.75): Test regular.
[0.75, 0.9): Test bueno.
[0.9, 0.97): Test muy bueno.
[0.97, 1): Test excelente.

iii) F1-Score: Es la medida de precisión que tiene un test. Se emplea en la determinación de un valor único ponderado de la precisión y la exhaustividad (recall).

iv) Precision: La precisión es el ratio entre el número de documentos relevantes recuperados entre el número de documentos recuperados. De esta forma, cuanto más se acerque el valor de la precisión al valor nulo, es decir nulo el valor del denominador, mayor será el número de documentos recuperados que no consideren relevantes. Si por el contrario, el valor de la precisión es igual a uno, se entenderá que todos los documentos recuperados son relevantes. Esta forma de entender la precisión introduce el concepto de ruido informativo y de silencio informativo.

v) Recall: Este ratio viene a expresar la proporción de documentos relevantes recuperados, comparado con el total de los documentos que son relevantes existentes en la base de datos, con total independencia de que éstos, se recuperen o no. Si el resultado de esta fórmula arroja como valor 1, se tendrá la exhaustividad máxima posible, y esto viene a indicar que se ha encontrado todo documento relevante que residía en la base de datos, por lo tanto no se tendrá ni ruido, ni silencio informativo: siendo la recuperación de documentos entendida como perfecta. Por el contrario en el caso que el valor de la exhaustividad sea igual a cero, se tiene que los documentos obtenidos no poseen relevancia alguna.

A continuación, se desglosan el promedio de los datos obtenidos post implementación del modelo de Adaboost, dependiendo de su % de distribución de remuestro, y sus métricas incluyen Accuracy, Area under the ROC curve, F1-Score, Precision y Recall:

	ronda	accuTest	rocTest	f1Test	precTest	recTest
	5	0,66838	0,61806	0,45296	0,47794	0,44934
% distrib	10	0,68829	0,63413	0,45357	0,53457	0,42523
10	15	0,69583	0,65102	0,51480	0,54199	0,50268
	20	0,70850	0,65817	0,52042	0,56016	0,50182

	ronda	accuTest	rocTest	f1Test	precTest	recTest
	5	0,69131	0,67636	0,64489	0,59673	0,71233
% distrib	10	0,70252	0,66955	0,60627	0,61940	0,60532
20	15	0,70552	0,69615	0,65682	0,60338	0,72608
	20	0,71138	0,70013	0,66138	0,61401	0,72500

	ronda	accuTest	rocTest	f1Test	precTest	recTest
	5	0,68006	0,66518	0,58275	0,55977	0,63695
% distrib	10	0,67979	0,64096	0,49016	0,56636	0,45724
30	15	0,67338	0,66054	0,56856	0,54700	0,62297
	20	0,68229	0,66780	0,57417	0,55242	0,63143

	ronda	accuTest	rocTest	f1Test	precTest	recTest
	5	0,66844	0,65805	0,62581	0,55968	0,71369
% distrib	10	0,67853	0,65902	0,59699	0,58700	0,62925
40	15	0,67451	0,66237	0,59965	0,57089	0,65654
	20	0,67866	0,66714	0,63430	0,56937	0,71982

Como se aprecia en la tabla anterior, el mejor resultado obtenido en cuanto a Accuracy, corresponde al de **71,1% en la ronda 20**, con 0,2% de distribución de remuestreo.

A continuación, se pueden observar las gráficas de Accuracy para las distintas distribuciones tomadas (desde 0.1 hasta 0.4 %). Como observación general, se aprecia que mientras mayor sea el % tomado, se tiene una mayor estabilidad mientras se avanza hasta la ronda 20.

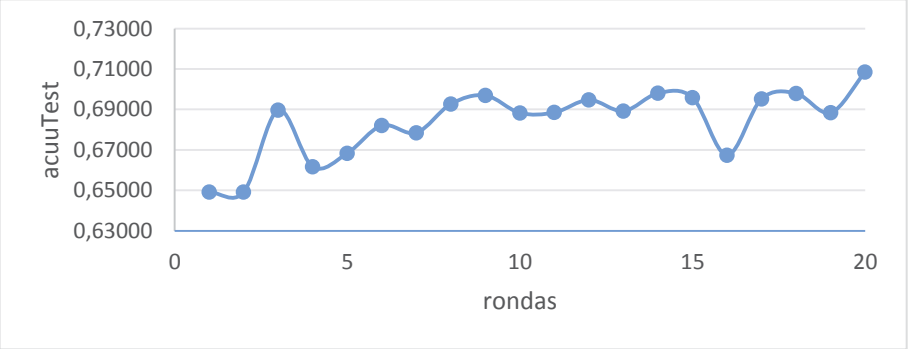


Figura 13, gráfico accuracy 0.1%

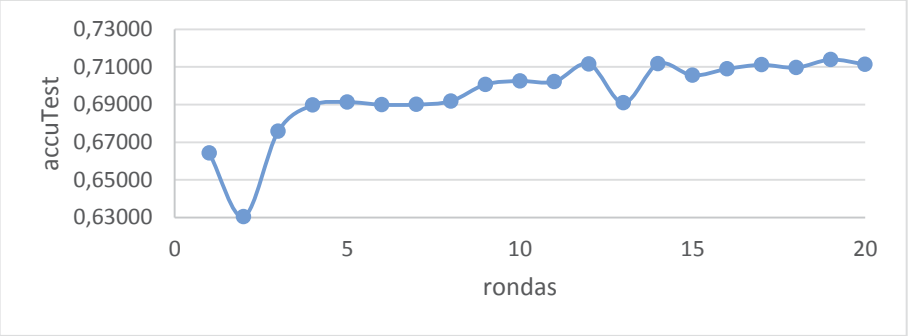


Figura 14, gráfico accuracy 0.2%

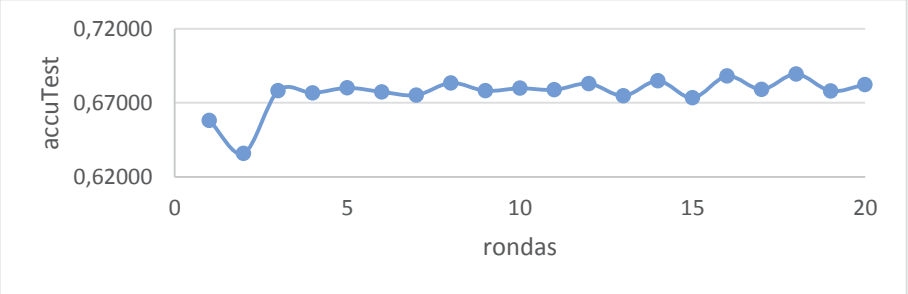


Figura 15, gráfico accuracy 0.3%

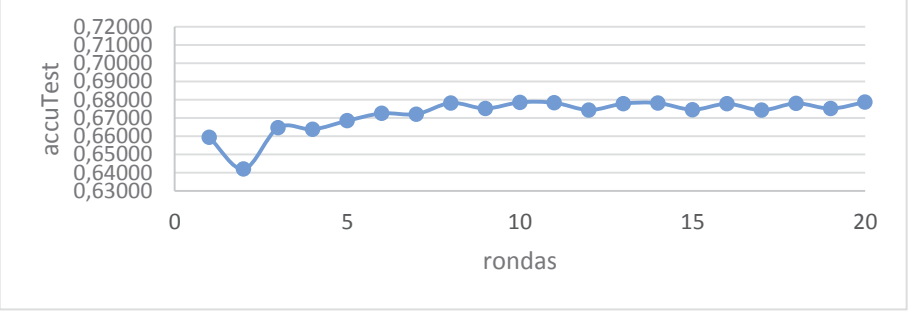


Figura 16, gráfico accuracy 0.4%

8 Comparativa de resultados

Como trabajo anterior y para poder tener una comparativa con otros modelos se presenta el paper de Felipe Oliva, de la Escuela de Informática de la Pontificia Universidad Católica de Valparaíso, el cual contiene como propuesta comparar el desempeño de diferentes clasificadores automáticos que son alimentado con mensajes realizados en la red social de Twitter con un dataset de tweets dentro del ámbito de la industria del retail. Esto se lleva a diferentes escenarios de entrenamiento (training) y de pruebas (testing).

Como datos previos, se tienen un conjunto de datos compuesto por 3161 mensajes clasificados de forma manual. Los tweets se encuentran en su totalidad en idioma español y pertenecen a las cadenas de Falabella y Ripley, son menciones tomadas entre Octubre de 2013 y Enero de 2014. El conjunto de datos se encuentra dividido en tweets positivos y negativos.

Para evaluar los resultados se utilizó tres métricas de rendimiento: la precisión, el recall y el valor F1, se mostrarán los resultados obtenidos por F1 ya que proporciona la medida más global de entre los tres anteriores. Recordemos que en el análisis estadístico de la clasificación binaria, la puntuación F1 es una medida de la precisión de una prueba. La puntuación F1 se puede interpretar como un promedio ponderado de la precisión y el recall, donde una puntuación F1 alcanza su mejor valor en 1 y peor puntuación a 0.

Se tiene como hipótesis: Si se clasifican marcas de un determinado rubro, como el retail, esto sirve para clasificar marcas que se encuentren dentro del mismo rubro (por ej: Con Ripley puedo clasificar Falabella, ya que están bajo el mismo rubro de negocio), expresado de mejor manera en la siguiente figura.

Figura 17, Hipótesis de prueba propuesta anterior

Se utiliza Adaboost con este mismo dataset del sector del retail, se pueden contraponer los siguientes resultados usando como regla principal que en ambos casos se utiliza un árbol de decisión para la clasificación de los textos cortos y se utiliza la representación TF-IDF en ambos casos. A continuación se presentan en la siguiente figura las diferencias.

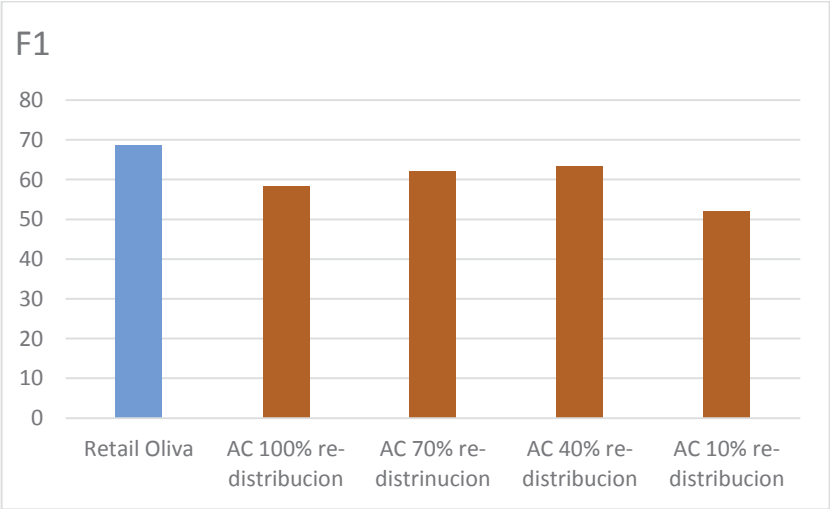


Figura 18, comparativa de resultados con trabajo anterior respecto a métrica F1

Como se puede observar en la figura anterior, el valor F1 del trabajo anterior llega a 68.6. A su derecha se diagraman los distintos valores de las pruebas con Adaboost Concurrente, teniendo como parámetro el cambio del porcentaje de redistribución entre las pruebas.

Así también, se estudia el comportamiento del algoritmo en cuanto al tiempo que demora en completar las 20 rondas, dependiendo del tipo de dataset que se vaya utilizando para su prueba. A continuación se presentan los distintos dataset ocupados en este proyecto y su respectivo promedio en segundos luego de haber realizado las 20 rondas, entregando un promedio de 6 experimentos, todos con un 40% de re muestreo.

Dataset	Tiempo (s) promedio luego de 20 rondas y 6 experimentos
Retail 3.161 tweets	22.05 s
Bachelet 5.000 tweets	25.31 s
Bachelet 10.000 tweets	43.07 s

Figura 19, tiempo luego de aplicación de algoritmo de Adaboost concurrente

Como se puede observar, existe una clara proporción respecto al tamaño del dataset respecto al tiempo utilizado en la aplicación del algoritmo de Adaboost concurrente.

9 Conclusiones

El trabajo presentado en este informe describe la problemática entorno a la clasificación de los mensajes de redes sociales como Twitter, como representar los mensajes y responder a los distintos escenarios que se plantearon, para luego poder inferir si estas opiniones corresponden a expresiones con polaridad positiva o negativas.

La etapa de clasificación manual es fundamental que se encuentre correcta, para que así los algoritmos puedan funcionar de manera correcta, por ello es que se trabajó con una mayor cantidad de personas para lograr mejores resultados.

La etapa de pre-procesamiento, al igual que en el proceso de minería de datos, forma un papel principal para la obtención de buenos resultados. Transformar los mensajes, realizar limpieza a los datos, convertirlos para que el clasificador entienda el texto es un punto fundamental.

La complejidad y riqueza de nuestros idiomas y de nuestro lenguaje nos permite hacer frases tremendamente complicadas, jugar con conocimiento implícito que no se refleja de forma explícita en nuestros actos de habla o, sencillamente, confiar en el contexto para que se entienda el significado real de lo que queremos decir, contexto que no tiene por qué estar disponible para el motor de análisis. Podemos detectar las ironías más evidentes, pero no las más sutiles, así los algoritmos vienen a alivianar de cierta medida esta complejidad, pero ninguno es 100% confiable.

Como conclusión del trabajo se puede decir que luego de utilizar el dataset con 10.000 mensajes de twitter de la candidata Bachelet, se logró la mejor performance un un 20% de distribución de remuestro llegando al 71.1%.

Como trabajo futuro, se puede señalar que la mayor complejidad de la aplicación de la solución fue el tratamiento de los datos y la creación del dataset, debido principalmente que al utilizar una mayor cantidad de tweets se crea un diccionario de palabras mayor, por lo que la matriz de representación crece de manera directa, generando problemas de manejo de memoria en el sistema y no pudiendo completar el algoritmo. Para esto se propone poder utilizar una representación distinta, pudiendo escalar en la cantidad de mensajes y evaluar la perfomance del mismo, pudiendo al tener un mejor entrenamiento mejores resultados.

10 Referencias

- Pang B. and Lee L.Opinion mining and sentiment analysis. Foundation and Trends in Information Retrieval, 2(1 -2):1 –135.
- Wiebe, J. M., Wilson, T., Bruce, R., Bell, M., and Martin, M. (2004). Learning subjective language. Computational Linguistics, 30:277 –308.
- Alfaro R., Allende H. (2010), Text Representation in Multi -label Classification: Two New Input Representations 10th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA'11).
- http://www.dc.uba.ar/materias/aa/2013/cuat1/index_html
- http://es.wikipedia.org/wiki/Aprendizaje_supervisado
- http://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo
- Análisis de Sentimientos sobre un Corpus en Español: Experimentación con un Caso de Estudio, Luciana Dubiau, Juan M Ale
- Aplicación de las máquinas de soporte vectorial para el reconocimiento de matrículas, Esther Gutiérrez Alonso, MADRID, Junio 2007
- Clasificación de documentos usando Maquinas de Vectores de Apoyo Martha Varguez Moo, Víctor Uc Cetina, Carlos Brito Loeza, 2012.
- Clasificación Automática del Sentido de los Mensajes de Usuarios de Twitter, Felipe Oliva Rodrigo Alfaro.
- http://en.wikipedia.org/wiki/F1_score
- http://aprendizajeincremental.blogspot.com/2011/08/aprendizaje-incremental-educacion.html
- SemiBoost: Boosting for Semi-supervised Learning Pavan Kumar Mallapragada, Student Member, IEEE, Rong Jin, Member, IEEE, Anil K. Jain, Fellow, IEEE, and Yi Liu, Student Member, IEEE.
- An AdaBoost Algorithm for Multiclass Semi-Supervised Learning Jafar Tanha, Maarten van Someren, Hamideh Afsarmanesh Informatics Institute, University of Amsterdam.
- Semi-Supervised Robust Alternating AdaBoost, Hector Allende-Cid, Jorge Mendoza, Hector Allende, Enrique Canessa.
- https://github.com/hrs/python-tf-idf
- https://es.wikipedia.org/wiki/Curva_ROC
- https://es.wikipedia.org/wiki/Precisi%C3%B3n_y_exactitud
- https://es.wikipedia.org/wiki/Precisi%C3%B3n_y_exhaustividad