INFOTEC CENTRO DE INVESTIGACIÓN E INNOVACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

DIRECCIÓN ADJUNTA DE INNOVACIÓN Y CONOCIMIENTO
GERENCIA DE CAPITAL HUMANO
POSGRADOS

"IDENTIFICACIÓN DE MENSAJES DE ALTA PRIORIDAD EN TWITTER PARA UNA INSTITUCIÓN FEDERAL"

SOLUCIÓN ESTRATÉGICA
Que para obtener el grado de
MAESTRO EN CIENCIA DE DATOS E INFORMACIÓN

Presenta:

Humberto García Flores

Asesor:

Dr. Leopoldo Altamirano Robles

Ciudad de México, 04 de Mayo de 2021

Autorización de Impresión

INFOTEC
POSGRADOS

AUTORIZACIÓN DE IMPRESIÓN Y NO ADEUDO EN BIBLIOTECA

MAESTRÍA EN CIENCIA DE DATOS E INFORMACIÓN

Ciudad de México, 16 de junio de 2021
INFOTEC-DAIC-GCH-SE-197/2021.

La Gerencia de Capital Humano / Gerencia de Investigación hacen constar que el trabajo de titulación intitulado

IDENTIFICACIÓN DE MENSAJES DE ALTA PRIORIDAD EN TWITTER PARA UNA INSTITUCIÓN FEDERAL

Desarrollado por el alumno **Humberto García Flores** y bajo la asesoría del **Dr. Leopoldo Altamirano Robles**; cumple con el formato de biblioteca. Por lo cual, se expide la presente autorización para impresión del proyecto terminal al que se ha hecho mención.

Asimismo se hace constar que no debe material de la biblioteca de INFOTEC.

Vo. Bo.

Lic. Juan Ramón Abarca Damián
Coordinador de Biblioteca

Anexar a la presente autorización al inicio de la versión impresa del trabajo referido que ampara la misma.

C.p.p Servicios Escolares

Agradecimientos

El logro de los objetivos y metas personales no sería posible sin el apoyo de un conjunto de personas que por razones académicas, laborales y/o emocionales contribuyeron y motivaron a la culminación de la tesis que aquí se presenta, por lo que externo a todos ellos mi sincero agradecimiento.

A los docentes, investigadores y cuerpo administrativo del INFOTEC, particularmente a la investigadora anónima designada a la revisión de este documento, por las finas observaciones que realizó y que contribuyeron a mejorar la calidad de la tesis.

A mi asesor el Dr. Leopoldo Altamirano Robles, por motivarme firmemente a continuar con los estudios de posgrado y por los atinados consejos para el desarrollo de este proyecto.

A mis amigos y compañeros en el Instituto Nacional de Astrofísica, Óptica y Electrónica y en el Instituto Tecnológico de Puebla que siempre estuvieron atentos de mí, me brindaron su apoyo incondicional y algunos de ellos sirvieron de voluntarios en el etiquetado de una base de datos esencial para este trabajo.

Y de manera especialmente significativa a mi familia nuclear y extendida, por el tiempo que estuve ausente. Que el esfuerzo realizado y aquí documentado sirva de ejemplo a mis hijos para que ocupen su tiempo en aprender y ser felices.

Tabla de contenido

Índice de figuras

Índice de gráficos

Índice de cuadros

Siglas y abreviaturas

csv. Formato de archivo electrónico de datos separados por comas.

.json. Formato de archivo electrónico de datos.

AMIPCI. Asociación Mexicana de Internet.

API. Interfaz de programación de aplicaciones (por sus siglas en Inglés), se trata de un conjunto de instrucciones compiladas para que puedan ser utilizadas por otro programa de computadora.

Corpus. Se refiere a un conjunto de mensajes que constituyen una base de datos para procesamiento de lenguaje natural.

DAI. Departamento de Análisis de Información.

hashtag. Es un tipo de etiqueta digital para dar seguimiento a un tema o persona particular.

Notebook Jupyter. Plataforma interactiva e informativa de desarrollo de software usando Python.

Python. Lenguaje de programación.

R. Entorno de desarrollo y lenguaje de programación con enfoque en el análisis estadístico.

SEPLN. Sociedad Española para el Procesamiento de Lenguaje Natural.

SVM. Máquinas de vectores de soporte.

TASS. Taller de Análisis de Sentimientos en la SEPLN.

TF. Frecuencia de Término en un documento.

TF-IDF. Frecuencia de Término - Frecuencia Inversa de Documento.

Trend topic. Tema que por su alcance social se ha convertido en una tendencia temática temporal

Twit. Mensaje escrito en la red social Twitter.

Twitter. Red social para compartir mensajes cortos y material audiovisual entre sus usuarios.

WIP. World Internet Project.

Introducción

Cada día se generan cientos de miles de mensajes breves en la red social Twitter, un microblog en el cual distintas personas mediante cuentas de usuario pueden compartir con los demás sus pensamientos y reflexiones (1), manifiestan opiniones e informan sobre hechos o eventos de su entorno, creando así una gran base de información colectiva que puede ser aprovechada con fines estratégicos para la formulación de políticas, sustentar la toma de decisiones, realizar análisis y otras muchas aplicaciones (2), de hecho las personas y sus organizaciones cada vez hacen un mayor uso de las redes sociales para consultar información que sirva de base para tomar decisiones (3).

Particularmente en el entorno político-institucional Twitter se ha convertido en una plataforma fundamental para el trabajo de políticos y periodistas, ya que les permite comunicar su trabajo, ya sean propuestas de campaña, políticas gubernamentales o notas periodísticas, a un público muy amplio que se sabe opinará al respecto dando origen a una comunicación digital pública que puede estudiarse a través de análisis de polaridad de sentimientos, estimación de intención de voto o el estudio de movimientos sociales y campañas digitales abruptas (4).

De acuerdo con Liu B. y Zhang L. (3) la minería de opinión se encarga del estudio de los elementos de la opinión, siendo ésta un sentimiento, actitud, emoción o valoración respecto a una entidad (producto, servicio, persona, evento, organización o tema) o un aspecto de la entidad y que es emitida por el titular de una opinión; uno de los elementos del estudio de opinión es el análisis de sentimientos el cual incluye entre sus aspectos a la polaridad, la cual tiene por objetivo identificar la orientación semántica de la opinión como positiva, negativa o neutral.

El análisis de polaridad puede abordarse desde tres puntos de vista: el uso de diccionarios o bolsas de palabras significativas que pueden incluir un valor de fuerza asociada a cada palabra, el aprendizaje computacional mediante algoritmos de clasificación que emplean conjuntos de entrenamiento previamente etiquetados, y una combinación de ambos métodos (3).

En el contexto antes descrito, el Departamento de Análisis de Información de una institución del Gobierno Federal[1] que coadyuva en la procuración de la paz y la seguridad interior de nuestro país, a la cual en lo subsecuente se le denominará DAI, con la cual colaboré en el desarrollo de proyectos tecnológicos mediante convenios interinstitucionales a través del INAOE (Centro Público de Investigación en el cual me desempeño actualmente), tiene la necesidad de emplear métodos automáticos para mejorar la eficacia en la identificación de mensajes prioritarios de Twitter, la cual es una actividad que realizan diariamente como parte de sus funciones preestablecidas. Motivado por lo anterior se propone el desarrollo de una solución tecnológica mediante una colaboración de naturaleza académica y experimental, lo cual respectivamente corresponde al desarrollo del presente trabajo de tesis y al producto de la tesis, el cual permitirá a la DAI realizar análisis experimentales para evaluar la pertinencia sobre proyectos futuros en esta materia.

Los mensajes prioritarios, de manera individual o en conjunto, son aquéllos que presentan características en su contenido que tiene potencial de afectar negativamente al DAI, a sus integrantes o a sus autoridades en el corto o mediano plazo. La afectación puede ser mediática o funcional, las afectaciones mediáticas están relacionadas con la opinión que se manifiesta hacia el DAI, mientras que las afectaciones funcionales tienen que ver con noticias o reportes de posibles actividades que implican amenazas al desempeño y que son identificables por una o más palabras de interés para el DAI, en ambos casos deben sugerirse acciones inmediatas.

En el presente trabajo se propone una solución estratégica para la identificación de mensajes prioritarios para el DAI empleando dos enfoques de análisis: el basado en diccionario y el basado en aprendizaje computacional.

El enfoque de diccionario se aborda mediante un modelo de discriminación de mensajes basado en la frecuencia de palabras significativas, las cuales se identificaron previamente e integraron en una bolsa de palabras a partir de una base de mensajes prioritarios del DAI.

[1] A solicitud de la institución no se menciona su nombre en este documento.

En el enfoque basado en aprendizaje computacional se hace uso del análisis de la polaridad de opinión de usuarios, el cual emplea clasificación supervisada, para tal efecto los clasificadores se entrenaron con una combinación de dos bases de mensajes en idioma español, etiquetados con una de tres clases: positiva, negativa y neutral. Una de las bases de mensajes fue elaborada por la Sociedad Española de Procesamiento de Lenguaje Natural (SEPLN) y la segunda fue elaborada como parte de este trabajo con la ayuda de 5 voluntarios que etiquetaron los mensajes, ambas bases se analizan en su desempeño como conjunto de entrenamiento y se concatenan en una sola para contar con un conjunto de entrenamiento que sea más representativo para clasificación supervisada, lo anterior con el objetivo de identificar si hay variaciones significativas respecto a los conjuntos de entrenamiento pues, debido a que la DAI evalúa distintos contextos que pudieran afectarle, se considera necesario contar con una base general de mensajes de entrenamiento que ofrezca una respuesta aceptable de precisión en cualquier contexto.

Capítulo 1

Planteamiento del proyecto

Capítulo 1. Planteamiento del proyecto

En este capítulo se decriben las motivaciones que dan origen al desarrollo del proyecto que se plantea en este documento como una solución estratégica.

1.1 Problemática

El DAI tiene entre sus funciones realizar el monitoreo de distintas redes sociales como Twitter, Facebook, Youtube e Instagram, para identificar mensajes que describan noticias, eventos, hechos u opiniones que pudieran afectar su desempeño o su imagen pública.

Particularmente para el monitoreo en Twitter, el DAI recupera mensajes que mencionan a la institución a la que pertenece, así como a sus principales funcionarios y autoridades, o a temas variados que consideran de interés prioritario para todos ellos.

Para comprender el proceso de trabajo, la problemática y las necesidades del DAI se sostuvo una conversación con el jefe del área, la cual está integrada por 3 analistas y 6 ayudantes además del jefe, de quien se obtuvo la información que a continuación se describe.

El proceso de trabajo empleado por el DAI para identificar mensajes prioritarios en Twitter consta de 3 pasos, los cuales se describen a continuación:

- Recopilación. Mediante clientes de TwitRSS recuperan hasta 1000 mensajes diarios que son almacenados en una base de datos MySQL. Esta cantidad está limitada por la tecnología de recuperación empleada y también porque no cuentan con personal suficiente para incrementar el volumen de análisis. La recopilación de mensajes se realiza conforme a dos criterios de búsqueda: la que menciona a la institución y cualquier otra que la DAI o la Institución considere pertinente conforme al contexto coyuntural actual, por lo que prácticamente deben recuperar y analizar mensajes relativos a cualquier tema: gobierno, funcionarios, política nacional e internacional, seguridad, empresas y empresarios, deportes, sanidad, finanzas, etc.
- Clasificación. Los mensajes recuperados diariamente son consultados y clasificados manualmente por una sola persona (analista) que los lee y determina si el mensaje es prioritario o no, sin que se asigne alguna etiqueta

de clase a cada tipo de mensaje. La prioridad del mensaje se analiza y discrimina por el analista en función de dos enfoques: la polaridad de su contenido (positiva, negativa o informativa) y si el contenido del mensaje incluye palabras significativas que describan eventos que podrían afectar el desempeño o imagen del DAI, de su institución o de los intereses que le sean conferidos como tarea de análisis.

- Decisión. Los mensajes identificados como prioritarios se guardan y asocian a casos de estudio para su atención, seguimiento y direccionamiento a otros departamentos con los que trabaja el DAI en la Institución.

Para los fines de esta tesis la DAI proporcionó una pequeña base de 256 mensajes prioritarios identificados conforme al criterio de contenido de palabras significativas, el cual se emplea en la etapa de análisis de esta tesis según se documenta en la sección 3.2.3.

El uso masivo de redes sociales y la cantidad de información que generan durante el día ha rebasado la capacidad diaria de análisis del personal en el DAI, resulta necesario mejorar su eficacia en la identificación de mensajes prioritarios en términos de cantidad y tiempo de procesamiento, siendo su indicador actual el de 1000 mensajes de Twitter analizados por día, según lo señalado por su personal durante dicha conversación.

La identificación de mensajes prioritarios en Twitter, entre miles que se generan diariamente, implican 2 retos: la estandarización de un procedimiento de trabajo para la adquisición, gestión y análisis de los mensajes, así como la identificación e implementación de métodos para el análisis automático de texto. Lo anterior con el propósito de reducir retrasos en la identificación de riesgos, hacer más eficaz el proceso de trabajo y estandarizar la selección de mensajes que actualmente se realiza en el DAI pues actualmente ésta depende del enfoque de cada analista.

La cantidad de mensajes que se analizan actualmente en el DAI está limitada por el medio de recuperación que se emplea (TwitRSS) y por la falta de métodos automáticos de análisis; si bien la primera limitante se resuelve empleando la API de

Twitter para desarrolladores[2], la segunda limitante tiene que ser abordada con técnicas de análisis de opinión que deben evaluarse para seleccionar las mejores en función de su desempeño (medidas de exactitud) y de su eficacia (cantidad de mensajes analizados respecto a tiempo de procesamiento).

En este trabajo se aborda una solución para mejorar la eficacia del DAI en la identificación de mensajes prioritarios, basada en técnicas para el análisis automático de la polaridad y en la identificación de palabras significativas.

1.2 Hipótesis

Dado que la Institución realiza un procedimiento de trabajo manual para identificar y analizar mensajes prioritarios con una eficacia de 1000 mensajes en una jornada laboral, el presente trabajo se desarrolla a partir de la siguiente hipótesis general:
¿En qué medida, empleando análisis de polaridad y frecuencia de palabras significativas, es posible mejorar la eficacia del proceso de identificación de mensajes prioritarios?

Considerando que los mensajes prioritarios son clasificados por el DAI en función de que su contenido refiere una opinión negativa o que describa un hecho que amenaza el desempeño de la institución, se plantean las siguientes hipótesis particulares para abordar ambos enfoques:

1. ¿Se puede mejorar la clasificación de la polaridad de mensajes al emplear la concatenación de bases de mensajes etiquetados como conjuntos de entrenamiento?
2. ¿En qué proporción pueden identificarse mensajes prioritarios a partir de palabras significativas contenidas en mensajes previamente calificados como prioritarios por el DAI?
3. ¿En qué medida la combinación de análisis de polaridad y de palabras significativas podría contribuir a una mejor identificación de mensajes prioritarios?

[2] La API de Twitter facilita la recuperación masiva de mensajes con ciertas restricciones de temporabilidad y periodicidad, disponible en: https://developer.twitter.com/en

Tomando en cuenta las anteriores hipótesis y buscando una respuesta a ellas, a continuación se describe el objetivo general y los particulares del presente trabajo.

1.3 Objetivos

En el desarrollo del presente trabajo se plantea el siguiente objetivo general y cuatro objetivos particulares.

1.3.1 Objetivo General

El objetivo general del trabajo es: analizar y evaluar algoritmos de clasificación para identificar eficazmente mensajes prioritarios de Twitter en función de su polaridad y de su contenido de palabras significativas así como la combinación de ambos.

1.3.2 Objetivos Particulares

A continuación se listan los objetivos particulares del trabajo, derivados del objetivo general:

.

A. Crear y concatenar bases de mensajes etiquetados que sirvan como conjuntos de entrenamiento para realizar análisis de polaridad.
B. Evaluar el desempeño de algoritmos de clasificación, aplicados al análisis de polaridad, considerando su eficacia en términos de cantidad de mensajes clasificados y tiempo de procesamiento.
C. Integrar un diccionario de palabras significativas a partir de mensajes prioritarios identificados previamente por el DAI.
D. Proponer e Implementar un método para la identificación de mensajes prioritarios empleando el diccionario de palabras significativas propuesto en el objetivo C.
E. Probar y analizar la coordinación de los enfoques de clasificación de análisis de opinión y de palabras significativas para la identificación de mensajes prioritarios.

1.4 Justificación

Aunque el DAI realiza monitoreo de distintas redes sociales, ha identificado que mejorar el proceso de análisis de mensajes prioritarios de Twitter puede

incrementar significativamente la eficacia de sus funciones por razones operativas y técnicas.

En primer lugar, el DAI ha identificado que la dinámica de Twitter es mayor que en otras redes sociales, esto es que la información se genera en mayor cantidad, de forma más rápida y con un alto potencial de propagación, de hecho así lo explica Murthy (1) al referir que "*Twitter brinda el espacio para que siempre haya una noticia permitiendo que una gran cantidad de personas se informe en un corto periodo de tiempo"*. Dada esta situación el DAI necesita incrementar su capacidad operativa para la recopilación y análisis de mensajes tal que le permite identificar riesgos oportunamente.

Twitter también presenta una ventaja técnica principal respecto a otras redes sociales, pues además de coleccionar un gran volumen de información, facilita herramientas (API de desarrollador) para la recuperación masiva de mensajes a partir de un criterio de búsqueda, motivando a que las personas y organizaciones realicen investigación diversa en el campo del análisis de texto.

Al incrementar la capacidad de recuperación de mensajes es necesario fortalecer y aumentar la capacidad de análisis, misma que debe estandarizarse y automatizarse.

La estandarización y automatización en el análisis de opinión es necesario y recomendable porque reduce la posibilidad de sesgos y aumenta la objetividad, pues como señalan Liu B. y Zhang (3), *"las personas dedican más atención a las opiniones que son afines a sus propias preferencias, se vuelven susceptibles a prejuicios subjetivos y presentan limitaciones mentales"* (debidas a razones biopsicosociales), *"produciendo resultados poco confiables cuando la cantidad de información a procesar es grande".*

Considerando lo anterior, la solución estratégica que se propone permitirá al DAI aumentar su capacidad de procesamiento, emplear métodos automáticos para estandarizar el análisis de opinión y aumentar la objetividad en la identificación de mensajes prioritarios.

1.5 Estructura de la Tesis

El presente trabajo está organizado de la siguiente manera: en el Capítulo 2 se describe el Marco Teórico del trabajo, incluye los conceptos que permiten comprender el contenido técnico y procedimental de la propuesta así como el Estado del Arte, donde se presenta la revisión de trabajos e investigaciones que están relacionadas con esta tesis. El Capítulo 3 describe la metodología empleada y su aplicación en el análisis de modelos para la clasificación de polaridad de mensajes de Twitter y la identificación de mensajes prioritarios usando un diccionario elaborado como parte de este trabajo. En el Capítulo 4 Resultados, se presentan la descripción y los resultados de aplicar el modelo de clasificación de la polaridad de mensajes de Twitter a dos conjuntos de 14,649 y 65406 twits respectivamente, el modelo de identificación de mensajes prioritarios basado en diccionario y un modelo híbrido aplicados a una base de 3769 twits.

Las conclusiones, aportaciones y oportunidades que resultan del desarrollo de la presente solución estratégica se presentan en el Capítulo 5.

Capítulo 2

Marco Teórico

Capítulo 2. Marco Teórico

La red social Twitter tiene un papel preponderante en la comunicación social de nuestros días, ha transformado nuestra manera de comunicarnos (4) y ha creado una base colectiva de información que las organizaciones y las personas emplean cada vez más para la toma de decisiones (2), esto ha dado origen a distintos enfoques para el estudio de mensajes como el análisis de opinión, la predicción de resultados electorales y el estudio de movimientos sociales abruptos (5). Poco después de su lanzamiento en 2006, Twitter mostró su éxito como herramienta de comunicación, propaganda y monitoreo de opinión en campañas electorales (6-12) y se ha identificado que existe una estrecha relación entre la carga de sentimientos vertidos en los mensajes de usuarios respecto a eventos sociales, culturales, políticos y económicos.

La carga de sentimientos en mensajes de Twitter es estudiado con la técnica denominada análisis de sentimientos, la cual incluye a la polaridad como uno de sus aspectos, es decir la orientación semántica de la opinión que puede clasificarse como positiva, negativa o neutral y se analiza empleando diccionarios, aprendizaje computacional o una combinación de éstos (3).

En esta sección se revisan y discuten trabajos de investigación que abordan el análisis de polaridad de sentimientos en mensajes de Twitter y su papel en el sector gubernamental, tal que permitan identificar los enfoques que han sido estudiados, los resultados obtenidos, los métodos de análisis y las técnicas empleadas, mismas que pueden ser de utilidad en la solución que se propone para mejorar la eficacia del proceso de identificación de mensajes prioritarios que afectan la imagen y el desempeño del DAI y de la institución a la cual pertenece.

Se plantea inicialmente una reseña de Internet y el papel social y político de Twitter, posteriormente se presenta una revisión de trabajos de investigación que han empleado el análisis de opinión en mensajes de Twitter con fines de gestión gubernamental, política y electoral, finalmente se describen las técnicas de análisis que se emplean en la propuesta de solución de este trabajo.

2.1 Twitter y su papel político y social

Es sabido en el contexto actual que, en la historia de la humanidad, son cuatro conjuntos de eventos los que han detonado el desarrollo industrial basado en ciencia y tecnología. Coloquialmente denominadas revoluciones industriales, la primera ocurrió a partir de la invención de la máquina de vapor en el siglo XVIII, la segunda se caracterizó por el desarrollo de la infraestructura energética de petróleo y electricidad, posteriormente vino el desarrollo de la industria electrónica que permitió llevar al hombre a la Luna, desarrollar el ordenador moderno y el internet. En la cuarta etapa de la revolución de la industria, todas las tecnologías que la humanidad ha creado facilitan la digitalización de procesos y la íntima relación entre máquinas, dispositivos y seres humanos.

En la tercera revolución de la industria sobrevino el desarrollo de Internet y no se ha detenido, siendo hoy una plataforma adoptada en el mundo, con más de 4,300 millones de usuarios (13), esto es el 57% de la población mundial.

Internet ha servido como base para el desarrollo de negocios soportados en aplicaciones WEB para comercio, información, comunicación, educación, entretenimiento y más. En materia de comunicación distintas aplicaciones permiten conectar a usuarios en redes sociales públicas y privadas, en las que se comparten contenidos de texto, imagen, audio y video.

Una de las aplicaciones de redes sociales más usada es Twitter, la cual de acuerdo al Reporte Global Digital 2019 (13) cuenta con 323 millones de cuentas en el mundo, de las cuales 4.3 millones son mexicanas, aunque se estima que sólo 2.5 millones son activas (14), considerando que la Asociación Mexicana de Internet (AMIPCI) y el World Internet Project (WIP) calculan que en México existen alrededor de 30 millones de usuarios de internet (idem), la penetración de Twitter en México es de aproximadamente el 13.5%.

Twitter, según Dhiraj Murthy (4) *"es un microblog en el cual los usuarios, en un corto periodo de tiempo, comparten entre sí pensamientos y reflexiones mediante mensajes breves"*, así mismo afirma que *"Twitter en relación a otras redes sociales como Facebook, es más público y social pues no pretende mantener*

lazos de amistad, sino acumular cada vez más seguidores que conocen el contenido publicado por un usuario".

Murthy manifiesta que *"Twitter es suficientemente amplio para alentar la apertura y la democracia, lo que le permite ser un medio persuasivo, equilibrado y significativo que debe ser tomado en cuenta"*, asimismo *"Twitter ha transformado la actividad periodística creando "periodistas ciudadanos" y brindando un espacio donde siempre hay una noticia, ha fomentado el activismo al servir como instrumento de comunicación para organizar movilizaciones"*, tal como ocurrió en las manifestaciones de la Primavera Árabe; señala también a Twitter como medio para alertar al mundo sobre desastres ya que permite que muchas personas estén al tanto de un evento de emergencia en muy poco tiempo. En resumen, Twitter ha modernizado las prácticas de comunicación al brindar una plataforma libre para que las personas compartan información, noticias, opiniones, reflexiones e ideas.

De acuerdo con Campos-Domínguez (5), Twitter *"ha tomado un papel preponderante en la comunicación política, ya que permite que los políticos hagan campaña, que los periodistas construyan notas y que el público comparta sus opiniones"* sobre las actividades políticas y las narrativas periodísticas, identifica además tres ámbitos de la comunicación política en Twitter: *"los emisores y receptores, el debate político y las campañas electorales"*.

De acuerdo con la autora, *"Twitter ha propiciado que los receptores del discurso político dejen de ser pasivos y compartan su opinión en tiempo real a través de esta red social sin importar donde se encuentren ni el medio por el cual reciben la información (radio, televisión, internet)"* y que puede estudiarse a través de análisis de polaridad de sentimientos, estimación de intención de voto o el estudio de movimientos sociales y campañas digitales abruptas como las revueltas en Egipto de 2011 y la Primavera Árabe en el periodo 2010-2012 (4).

En materia electoral según Tumasjan et al. (6) *"el éxito de las redes sociales en la elección presidencial de los Estados Unidos en 2008 hizo de Twitter (y otras redes) una herramienta integral para las campañas, facilitando el discurso político y el debate entre los electores, quienes manifiestan en sus mensajes una estrecha carga sentimental respecto a su preferencia electoral",* así ocurrió en la

elecciones estadounidenses (7), en las alemanas de 2009 (6), en las elecciones generales de Irlanda en 2011 (8 y 9) y de manera similar en otras (8) como las de Francia en 2012, en Italia en 2013, en Brasil en 2014 (10), en Reino Unido en 2015 (11) y en Austria en 2016 (12).

Sin duda, conforme a lo antes descrito, las opiniones que se manifiestan en Twitter han modificado la práctica de la comunicación humana y están transformando algunas de las actividades en la esfera gubernamental y política, los usuarios manifiestan opiniones e informan sobre hechos o eventos de su entorno, creando así una gran base de información colectiva que puede ser aprovechada con fines estratégicos para la formulación de políticas, sustentar la toma de decisiones, realizar análisis económico y otras aplicaciones (2), de hecho *"cada vez más personas y organizaciones utilizan las opiniones publicadas en redes sociales para la toma de decisiones"* (4).

De acuerdo con Liu y Zhang (3) *"la minería de opinión se encarga del estudio de los elementos de la opinión, siendo ésta un sentimiento, actitud, emoción o valoración respecto a una entidad (producto, servicio, persona, evento, organización o tema) o un aspecto de la entidad y que es emitida por un titular de opinión";* uno de los elementos del estudio de opinión es el análisis de sentimientos el cual incluye entre sus aspectos a la polaridad, la cual tiene por objetivo identificar la orientación semántica de la opinión como positiva, negativa o neutral.

Considerando que el DAI analiza el contenido de mensajes en Twitter para identificar como prioritarios aquéllos que por su contenido podrían afectar su imagen o su desempeño y direccionarlos a otras áreas para la toma de decisiones, a continuación se revisan y discuten trabajos de investigación que abordan el análisis de opinión en torno a mensajes emitidos por instituciones de gobierno o relativos a éstas, a las políticas o acciones implementadas así como al análisis de mensajes durante procesos electorales, tal que permitan identificar técnicas y procedimientos empleadas en la solución de problemas similares al que se plantea.

2.2 Análisis de sentimientos en Twits de Gobierno

Considerando que el DAI pertenece a una institución federal y que realiza el análisis de mensajes para identificar riesgos que puedan afectar su imagen (polaridad de opinión) o su desempeño (intención de acción), en este apartado se realiza la revisión de trabajos de investigación relativos al análisis de polaridad de sentimientos en mensajes de twits relacionados con gobierno y política conforme a dos enfoques, el análisis de intención basada en sentimiento y el análisis de respuesta basada en sentimiento conforme a dos casos previamente investigados por distintos autores: la intención del voto en procesos electorales y la respuesta de usuarios a políticas gubernamentales.

2.2.1 Análisis de intención basada en sentimiento (caso: procesos electorales)

Uno de los temas de mayor interés, por la gran participación y audiencia que implica, es el análisis de mensajes en procesos electorales, esto con dos propósitos principales: predecir el resultado de las elecciones a partir de la opinión de los usuarios así como comparar los resultados respecto a encuestas tradicionales; la premisa fundamental de estas investigaciones, y por lo cual se aborda su revisión en este trabajo, es que la orientación semántica de los mensajes puede ser reflejo de la intención de llevar a cabo una acción (en este caso intención de voto), ya sea en masa o en lo individual. Identificar esta intención es útil en el análisis de riesgos cuando ha sido definido previamente un sujeto o evento de interés. Motivado por lo antes descrito, a continuación se revisan y comentan algunas investigaciones de análisis de mensajes de Twitter relativas a procesos electorales en el mundo.

Diakopoulos y Shamma (7) realizaron una de las primeras investigaciones sobre el análisis de sentimiento en twits de orden político-electoral. Derivado del debate presidencial de Estados Unidos en 2008 recopilaron 3200 mensajes de Twitter y los clasificaron manualmente como positivo, negativo, mixto u otro. Para dicha clasificación manual emplearon razonamiento humano contratado a terceros y después realizaron una validación para evaluar la calidad del etiquetado, posteriormente realizaron un análisis de sentimientos en mensajes agrupados

según el horario asociado a la participación de cada candidato en el debate. Si bien no emplean algún tipo de clasificación automática, vislumbran que su uso puede resultar muy eficaz a partir del conjunto de datos que etiquetaron meticulosamente.

Tumasjan, et al. (6) presentan un estudio de análisis de sentimientos realizado con poco más de 100 mil twits que mencionan a los candidatos de las elecciones federales de Alemania en 2010, para tal efecto usaron el software comercial LIWC2007 (Linguistic Inquiry and Word Count), el cual emplea un diccionario para determinar la polaridad y emociones manifestadas en los mensajes. A partir de los resultados los autores concluyen que Twitter es un indicador de los sentimientos políticos en tiempo real, que los mensajes reflejan el resultado de la elección e incluso se acercan a los resultados de las encuestas electorales tradicionales.

Este artículo es uno de los primeros trabajos donde se presentan resultados que concluyen que la polaridad y las emociones en mensajes de Twitter mantienen una relación significativa con los resultados de un proceso electoral, es decir que la intención generalizada de la decisión del voto está relacionada con la manifestación emocional en mensajes de Twitter, sin embargo presenta algunas limitaciones que deben tenerse en cuenta como el uso de una sola técnica de clasificación (diccionario) y el empleo de una herramienta de software no diseñada por los investigadores lo cual puede limitar el alcance de los análisis.

Para las elecciones generales de Irlanda en 2011 Bermingham y Smeaton (8) recopilaron más de 32 mil twits relativos a 5 partidos políticos, a los cuales realizaron análisis de polaridad empleando algoritmos de clasificación supervisada y descartando el uso de aprendizaje no supervisado combinado con diccionarios debido a que en experimentos previos obtuvieron resultados menos precisos. Para generar sus conjuntos de entrenamiento, emplearon a 9 personas que etiquetaron distintos mensajes conforme a su orientación semántica (positivo, negativo y neutral), lo anterior mediante sesiones de discusión conjunta. Los algoritmos que emplearon para la clasificación así como la exactitud promedio obtenida fue la siguiente: Máquinas de Vectores de Soporte 64.82%, Multinomial Naive Bayes

62.94% y Adaboost Multinomial Naive Bayes 65.09%. Los mensajes fueron representados con vectores generados con TF-IDF. Los autores compararon la clasificación de la polaridad expresada hacia los partidos en Twitter con las encuestas tradicionales y los resultados finales de las elecciones, encontrando que sus resultados presentaron un margen de desviación que debe ser revisado en el futuro para mejorarse. En este trabajo resulta de interés el empleo de criterio humano para el etiquetado de los mensajes que constituyen los conjuntos de entrenamiento, mismos que a partir de discusiones de grupo clasificaron manualmente, sin embargo no especifican el tamaño de los conjuntos, ni los tipos de mensajes empleados ni la reglas ocupadas para su clasificación, esta información podría ser de gran utilidad para explicar los índices de precisión obtenidos e incluso mejorarlos, más aún cuando comentan que la técnica de aprendizaje no supervisado combinada con diccionarios es menos precisa.

Bakliwal, Foster, et al. (9) realizan análisis de sentimientos a 2624 mensajes de Twitter recopilados durante la jornada previa a las elecciones generales de Irlanda en 2011, analizan 3 enfoques con el objetivo de identificar la alternativa de clasificación más precisa. El primer enfoque es la estimación subjetiva de la polaridad basado en la combinación de dos diccionarios: Subjectivity Lexicon (SL) (15) y SentiWordNet (16); el segundo enfoque es el etiquetado manual de sentimientos y su clasificación empleando aprendizaje supervisado con máquinas de vectores de soporte y validación cruzada dividiendo la base en 5 conjuntos; el tercer enfoque es la combinación de los dos primeros, es decir, se ocupan como características de los conjuntos de datos el etiquetado manual y la puntuación de polaridad obtenida con los diccionarios. Para etiquetado manual emplearon a 3 especialistas en el ámbito político, 2 de ellos para asignar a cada mensaje 1 de 3 etiquetas y un tercer especialista para desempate en caso de requerirse. Los especialistas se encargaron de eliminar mensajes sarcásticos. La exactitud obtenida para el análisis usando diccionarios es de 59%, para la clasificación supervisada a partir de etiquetado de expertos es de 55%, y la combinación de los diccionarios y el etiquetado manual para integrar un conjunto de entrenamiento presenta una ligera mejora al alcanzar una exactitud de 62%.

Silva, et al. (10) presentan un estudio donde identifican que el análisis de polaridad de sentimientos en Twitter puede ser útil para conocer las preferencias de votantes a partir de sus opiniones. Compararon el análisis de polaridad de sentimientos con los resultados de seis encuestas tradicionales de intención (opinión positiva) y rechazo (opinión negativa) del voto, observaron que se generaron aproximadamente los mismos resultados con una diferencia respecto a estas de 1% a 8%. Para realizar el análisis de polaridad colectaron 92 mil twits relativos a las elecciones brasileñas de 2014, clasificaron manualmente (sin especificar cómo lo hicieron) un conjunto de entrenamiento a partir de una muestra aleatoria simple del 30% del total de los datos, finalmente emplearon una licencia del software DiscoverText para realizar clasificación supervisada. Los investigadores concluyen que el análisis de sentimientos en Twitter es una técnica de bajo costo respecto a las encuestas de opinión tradicionales, sobre todo cuando se realiza a gran escala como en unas elecciones presidenciales, pero no ha alcanzado el mismo nivel de precisión que las encuestas de opinión tradicional.

Para las elecciones generales de Reino Unido en 2015 Burnap, Gibson, et al. (11) recopilaron más de 32 millones de mensajes de Twitter, los clasificaron manualmente por partido político o candidato y empleando el software SentiStrengh de Thelwal, et al. (17) realizaron análisis de polaridad de sentimientos, dicho software emplea un diccionario que valúa cada palabra entre -5 y +5 en función de su fuerza y orientación semántica. Posteriormente comparan las proporciones de mensajes positivos con los resultados de la elección, observando desviaciones significativas que atribuyen a la ausencia de un análisis regionalizado pues se sabe existe la presencia de partidos mayoristas en ciertas regiones. Este artículo emplea el enfoque de frecuencia de palabras de un diccionario generalizado en idioma inglés para determinar la polaridad de mensajes en Twitter, de los resultados que obtienen se observa que las proporciones de la polaridad positiva no son reflejo preciso de la preferencia de voto debido a que los resultados se sesgan donde suponen existe una población que favorece mayoritariamente a algún partido político. El análisis puede emplearse con cierta cautela en la toma de decisiones previo a la fecha de la

elección cuando podría ser más probable que el sentimiento corresponda a la intención de actuar.

Durante el proceso electoral federal de Austria en 2016, Kušen y Strembeck (12) recopilaron más de 343 mil mensajes de Twitter, a los cuales realizaron análisis de polaridad de sentimientos empleando un proceso estructurado de 4 fases a partir de definir sus preguntas de investigación: 1) extracción, 2) preprocesamiento, 3) preparación y 4) análisis. Para la extracción emplearon la API de Twitter, en el preprocesamiento realizaron limpieza del texto para eliminar signos de puntuación y palabras sin orientación semántica, en la fase de preparación realizaron la valoración de polaridad empleando el software SentiStrengh de Thelwal et al. (17) el cual, como ya se comentó tiene un enfoque basado en diccionario, finalmente en su fase de análisis discuten la polaridad de los mensajes respecto a los resultados de las elecciones, observando que el candidato ganador mantuvo una postura más neutral en sus mensajes respecto a su contendiente. Este artículo proporciona un estructura ordenada para realizar análisis de polaridad de mensajes de Twitter, conduciendo sus resultados a una discusión que permite explicar la contienda electoral desde la ventana virtual que brinda Twitter, dicho método puede ser adoptado para realizar análisis de polaridad independiente del enfoque técnico empleado para su identificación.

2.2.2 Análisis de respuesta basada en sentimientos (caso: gestión gubernamental)

El análisis de opiniones de los usuarios de Twitter en relación a las acciones o políticas implementadas por el gobierno es un tema de interés para distintos investigadores ya que se considera pueden ser de gran utilidad para obtener retroalimentación no solicitada sobre el desempeño de dichas acciones y de los funcionarios; la premisa fundamental de las investigaciones sobre la opinión de usuarios respecto a la gestión gubernamental, y por lo cual se aborda su revisión en este trabajo, es que la orientación semántica de los mensajes relativos a políticas públicas o acciones de gobierno puede reflejar la aprobación de los ciudadanos respecto a cada una de éstas o respecto al desempeño de los funcionarios, tal que permita reorientar o fortalecer estrategias de gobierno.

Considerando lo antes descrito, a continuación se revisan y comentan algunas investigaciones de análisis de mensajes de Twitter relativas a la gestión de algunos gobiernos en el mundo.

Chen, Franks y Evans (18) refieren que el estudio de las redes sociales ha ganado notoria importancia para conocer la opinión de los usuarios respecto a algún tema en particular, lo cual desempeña un papel importante para influir y hacer crecer la relación entre el gobierno y los ciudadanos. Presentan un trabajo comparativo de análisis de polaridad de sentimientos en mensajes de Twitter de cuentas empleadas por 20 gobiernos de ciudades de Estados Unidos y Canadá, los autores recopilaron las respuestas de los usuarios a distintos mensajes generados desde cuentas gubernamentales relativos a información de servicios o seguimiento a solicitudes ciudadanas, para lo cual emplearon tres aproximaciones: análisis basado en la combinación de 3 diccionarios preexistentes en idioma inglés, análisis con aprendizaje computacional empleando un conjunto de entrenamiento preexistente que consta 1.6 millones de mensajes etiquetados para 3 clases balanceadas (positivo, negativo y neutral) y un enfoque adicional usando el método denominado SentiStrength; los autores refieren que aunque las tres técnicas son diferentes, sus resultados son estadísticamente robustos y comparables, agregan que el análisis de sentimientos presenta una estrecha relación con el contexto que se analiza y que las diferentes interpretaciones de los datos gubernamentales de las redes sociales y de la participación ciudadana en las redes sociales puede ser útil para la formulación de políticas y mejorar la confianza pública.

En el trabajo de investigación referido, para la clasificación los autores emplean diccionarios preexistentes en idioma inglés y que han sido probados previamente por otros autores; de acuerdo a los resultados que presentan esto es una ventaja pues no deben realizar trabajo previo para desarrollar conjuntos de entrenamiento, sin embargo, variaciones en el contexto de los mensajes puede afectar el desempeño de los clasificadores, por lo que esta desventaja debe tenerse en cuenta en el diseño de los conjuntos de entrenamiento.

Watequlis y Puspitasari (19) presentan un trabajo de análisis de polaridad de sentimientos en textos de Twitter en idioma indonesio empleando clasificación supervisada. Los textos son recuperados usando la API de Twitter, preprocesados para eliminar signos de puntuación, números, URLs, retwits, espacios innecesarios y mensajes repetidos; posteriormente clasifican manualmente cada mensaje como positivo o negativo. Las listas de mensajes positivos y negativos se dividen en 75% para entrenamiento y 25% para prueba; extraen las características de las palabras y emplean el clasificador Naïve Bayes de NLTK. Sus resultados muestran un 66% de exactitud para el conjunto de prueba. El trabajo fue empleado para analizar las reacciones de usuarios de Twitter a las políticas presentadas por el gobierno a través de dicha red social. Los investigadores manifiestan que el uso de lenguaje informal (jerga) hace complicado el etiquetado del conjunto de entrenamiento (aunque no comentan cómo lo realizan) y afecta el resultado de los clasificadores. Considerando lo anterior debe tenerse especial cuidado en la selección del contenido de los mensajes del conjunto de entrenamiento para que éste incluya palabras de connotación positiva y negativas que sean representativas para la clasificación de los mensajes.

Pradany y Fatichah (20) presentan una investigación sobre análisis de polaridad respecto a las políticas del gobierno indonesio publicadas en Twitter, para lo cual plantean tres desafíos: a) la ausencia de una estructura estándar para los mensajes; b) el contenido de los mensajes es amplio y heterogéneo, lo que dificulta la clasificación de temas y de sentimientos; y c) el lenguaje informal puede interpretarse de manera ambigua y no estar incluida en diccionarios. Una vez que realizan el proceso de limpieza de mensajes recuperados efectúan una clasificación no supervisada empleando K-medias, esto con el objetivo de identificar grupos de mensajes por tema y seleccionar aquéllos de interés para el tema que se analiza; posteriormente realizan el etiquetado manual de cada mensaje asignando la calificación de positivo, negativo o neutro y aplican el método denominado SentiWordNet (16) para generar un conjunto de entrenamiento para clasificación supervisada; finalmente entrenan un modelo de Máquinas de Vectores de Soporte para clasificar mensajes futuros con alguna de

estas tres calificaciones. Los autores manifiestan que una de las principales limitantes es el etiquetado manual que consiste en traducir los textos del conjunto de entrenamiento de indonesio a inglés conforme a los estándares de SentoWordNet, manifiestan la necesidad de un trabajo futuro para elaborar el etiquetado en idioma local y sea menos susceptible al lenguaje informal.

Una de las principales desventajas técnicas y operativas de este trabajo es que, debido el uso de una herramienta para análisis de opinión que ha sido desarrollada y optimizada para idioma inglés, deben traducir cuidadosamente a éste los mensajes en idioma local, más aún con el uso de la jerga local que primero debe interpretarse con palabras formales y luego traducirse también. Si bien los autores en su investigación afrontan los 3 desafíos planteados para el análisis de sentimientos, existe un cuarto que tiene que ver con la traducción del lenguaje, el cual de no tratarse cuidadosamente puede afectar la eficacia con la que han sido abordados los 3 desafíos.

Silva, et al. (21) realizaron un estudio de análisis de opinión en mensajes de Twitter respecto a cuatro programas sociales en Brasil, donde identificaron que el análisis de polaridad puede reflejar la opinión pública acerca de dichos programas promovidos por el gobierno y contribuir en la gestión social en el contexto de la estrategia de redes. La gestión social, conforme a lo descrito por los autores, se refiere al uso de los flujos de comunicación para convertirlos en acciones y decisiones políticas, y la estrategia de redes se concibe como una herramienta estratégica para compartir información y generar conocimiento a través de la interacción y discusión de problemas reales con los ciudadanos; considerando lo anterior los autores estiman que las opiniones realizadas por los ciudadanos en masa podrían contribuir a cambiar las decisiones de gobierno respecto a políticas emitidas. Para realizar el estudio recopilaron 10 mil mensajes de 4 programas sociales, tomaron una muestra del 30% y los clasificaron manualmente como positivo, negativo o neutral sin especificar qué reglas, normas o prácticas emplearon para esta tarea, posteriormente emplearon una licencia del software DiscoverText para entrenar un clasificador para cada uno de los cuatro conjuntos de mensajes y clasificar el resto de los mensajes, a partir de una muestra aleatoria

del conjunto clasificado determinaron precisiones de 74% a 86%. Como parte del estudio identificaron una limitante orgánica y otra técnica, la primera es que las opiniones tienen una vigencia temporal y no son necesariamente de personas que se benefician de los programas que se analizan y por otro lado, los mensajes con contenido irónico son difíciles de identificar y afectan la precisión de la clasificación de polaridad de sentimientos. Este artículo se enfoca mayoritariamente en las bondades del análisis de polaridad para la gestión social y en menor medida en las técnicas empleadas; los autores realizan un profunda reflexión sobre el papel democrático e independiente de la participación ciudadana reflejado en las redes sociales, en las cuales es necesario invertir para la transformación de datos en conocimiento útil para su aplicación en políticas públicas que beneficien a la sociedad.

Ceron y Negri (22) realizaron un análisis de polaridad de sentimientos en Twitter relativos a la Reforma Comercial y a la Reforma Educativa de Italia en 2015 y lo explican en función del discurso adoptado, a partir de esto determinaron que las opiniones públicas no solicitadas pero compartidas en redes sociales pueden ayudar en la elaboración de políticas públicas. Para realizar el análisis de opinión recopilaron más 730 mil twits y emplearon el método denominado Análisis de sentimientos agregado supervisado (SASA, por sus siglas en inglés), el cual consta de dos pasos: primero generar un conjunto de mensajes etiquetados por personas como positivo o negativo, que sea homogéneo respecto al conjunto de textos a evaluar, y segundo, emplearlo para entrenar un modelo de aprendizaje supervisado para clasificación automática de nuevos mensajes. Los autores manifiestan que el etiquetado humano es más efectivo que los enfoques basados en diccionarios, sobre todo cuando los mensajes contienen variaciones e interpretaciones del lenguaje, discurso irónico o refieren un contexto o tema distinto al que se analiza. Los autores señalan que el análisis de polaridad en Twitter facilita la toma de decisiones en política pública de tres maneras: proponer alternativas en función de las preferencias identificadas, identificar actores clave y procesos de movilización, monitorear el comportamiento de los usuarios durante la implementación de las políticas; indican también que Twitter es una alternativa

barata de análisis que enriquece el aprendizaje gubernamental pero no sustituye a los métodos tradicionales de investigación.

Hopkins y King (23) proponen un método basado en estadística no paramétrica para el análisis de textos, el cual permite estimar sin sesgo las proporciones de las categorías de interés para un conjunto de documentos, sin realizar categorización por documento e incluso cuando la precisión del clasificador individual (por documento) no es buena. Para probarlo emplearon miles de opiniones relativas a la presidencia de Estados Unidos. Para generar un primer conjunto de datos de entrenamiento realizan etiquetado manual para 6 categorías de documento; la investigación incluye un estudio del error del resultado en función del tamaño del conjunto de datos etiquetado manualmente, donde destaca que si el conjunto crece de 200 a 1000 documentos etiquetados el error se reduce en dos unidades porcentuales.

2.2.3 Discusión sobre el análisis de sentimientos en twits de gobierno

En los párrafos anteriores se ha presentado la descripción de algunos trabajos relevantes para esta tesis, los cuales consideran la importancia y el papel de Twitter como medio de comunicación social, político y gubernamental, así como el análisis de polaridad de sentimientos que ha sido empleado para evaluar la opinión de los usuarios de la red social en materia de políticas públicas, monitoreo de protagonistas y eventos, así como intención de voto en campañas electorales, lo anterior como un instrumento útil para aprovechar el conocimiento colectivo en el análisis de riesgo, la toma de decisiones y la mejora de estrategias de gobierno en la implementación de políticas y acciones.

En el conjunto de trabajos revisados se identifican 3 enfoques de análisis de la polaridad: basado en diccionario, clasificación con aprendizaje computacional y enfoque híbrido combinando diccionario y aprendizaje. De los trabajos revisados solo uno aborda el problema de análisis de polaridad con un enfoque híbrido, Bakliwal, Foster, et al. (9), en el cual se propone la integración en uno de dos diccionarios existentes, para etiquetar mensajes en idioma inglés de un conjunto de entrenamiento para clasificación con aprendizaje supervisado. El enfoque de diccionario es abordado por Tumasjan, et al. (6), Kušen y Strembeck

(12) y Chen, Franks y Evans (18). El enfoque de clasificación, en su mayoría basado en aprendizaje supervisado es abordado en la mayoría de las investigaciones revisadas (7, 8, 9, 10, 18, 19, 20, 22, 23).

En el cuadro 1 se muestra un resumen de los trabajos revisados, ordenados conforme al enfoque de análisis que emplean (diccionario, aprendizaje computacional o híbrido) y describiendo las consideraciones técnicas de los autores.

Enfoque Diccionario	**Enfoque Aprendizaje**	**Enfoque Híbrido**
Uso de software comercial con enfoque de valoración usando diccionario con fuerza de sentimiento para cada palabra. (6) Uso de software comercial con enfoque de valoración usando diccionario con fuerza de sentimiento para cada palabra, aplicado en el análisis del discurso de los candidatos a primer ministro en Austria (12) Emplean dos enfoques de diccionario, el primero es una colección de 3 diccionarios preexistentes en idioma inglés con los cuales realizan clasificación cuantitativa conforme a las palabras de cada mensaje, el segundo enfoque es usando software comercial que cuenta con su propio diccionario. Lo	Realizan clasificación basada en razonamiento humano y sientan uno de los primeros precedentes para realizar análisis automático de sentimientos en twits políticos. (7) Realizaron etiquetado manual con criterio humano para entrenar modelos de clasificación automática con índice TF-IDF. (8) Realizaron etiquetado manual con criterio humano para entrenar clasificadores de polaridad de mensajes sobre preferencia electora en Brasil y compararlo con encuestas tradicionales, refieren variación significativa en la precisión respecto a éstas (10). Etiquetan manualmente el conjunto de entrenamiento para clasificar la polaridad de la opinión respecto a políticas del gobierno indonesio, se reporta que el lenguaje informal (jerga) afecta la precisión de la clasificación (19). Realizan clasificación supervisada de la polaridad en mensajes que contienen la opinión de usuarios respecto a políticas de gobiernos de E.U. y Canadá, para lo cual emplean una base de mensajes etiquetados	Combinación de 2 diccionarios elaborados previamente combinados a un conjunto de mensajes etiquetados como positivo, negativo y neutral por 3 expertos en política, ambos para integrar una base de mensajes de entrenamiento con los cuales realizan clasificación no supervisada (11).

anterior para evaluar la opinión de usuarios respecto a políticas de gobiernos de E.U. y Canadá. (18)	preexistente como conjunto de entrenamiento (18). Clasifican polaridad de mensajes respecto a opinión de políticas implementadas por el gobierno indonesio; emplean una base de mensajes etiquetada manualmente previamente traducida al idioma inglés como conjunto de entrenamiento para su uso en el software SentoWordNet (20). Realizan clasificación de polaridad en mensajes relativos a 4 programas sociales en Brasil. Realizan etiquetado manual basado en criterio humano para un conjunto de entrenamiento. Refieren limitaciones para clasificar el lenguaje irónico así como la vigencia de la polaridad (21). Realizan clasificación de polaridad de opiniones en relación a programas del gobierno de Italia. Para el conjunto de entrenamiento realizan el etiquetado de mensajes con criterio humano para 2 clases: positivo y negativo. Refieren que es más efectivo que el uso de diccionarios sobre todo cuando los mensajes contienen variaciones e interpretaciones del lenguaje, ironía o variaciones de contexto (22). Se refiere que el uso de una mayor cantidad de ejemplos en el conjunto de entrenamiento reduce el error en la clasificación (23).	

Cuadro 1. Enfoques del análisis de polaridad abordados por distintos investigadores. Fuente: elaboración propia

A partir de la revisión de los artículos antes descritos se identifican aspectos importantes a tener en cuenta:

1. El análisis de polaridad de sentimientos en mensajes de Twitter puede reflejar la intención de los emisores para tomar una decisión, por ejemplo emitir su voto a favor de un candidato o participar en algún evento (6, 11, 10,12).

2. El análisis de polaridad puede ayudar a identificar la opinión de la sociedad (a través de los usuarios en redes sociales) respecto al desempeño de políticas de gobierno, por lo que puede resultar de utilidad para tomar decisiones en materia de gestión gubernamental y mejorar la confianza pública en las instituciones, esto sin que sustituyan a otros estudios tradicionales y que, aunque es relativamente económico es importante que se realice la inversión para

convertir información colectiva en conocimiento útil para tomar decisiones (9, 18, 19, 20, 21, 22).

3. No todos los mensajes que se recopilan sobre algún tema, sujeto o evento contienen información de interés, es importante entonces considerar procesamiento previo al análisis de polaridad para discriminar mensajes cuyo contenido podría no ser relevante para el análisis (20).

4. El enfoque de análisis de polaridad empleando clasificación supervisada presenta mejor desempeño cuando en los conjuntos de prueba y entrenamiento se emplean mensajes en el mismo contexto (19).

5. El etiquetado manual de la polaridad, basada en criterio y experiencia de humanos, tiene mejor resultado en el aprendizaje supervisado que el etiquetado empleando diccionarios (9, 10, 12, 19, 20, 21, 22), sobre todo cuando se emplean traducciones de dichos diccionarios a idioma local (17,19, 20).

6. La combinación de distintas bases de mensajes etiquetados pueden mejorar el desempeño de clasificadores (9), además que al crecer el conjunto de entrenamiento el error de clasificación se reduce (23).

Descrito lo anterior, el análisis de polaridad de sentimientos es una herramienta útil en la gestión gubernamental; la opinión positiva, negativa o neutral de un usuario o conjunto de usuarios puede ser reflejo de su intención de actuar en favor o no de una política pública, por lo que su identificación en el círculo gubernamental es prioritaria para tomar decisiones e implementar nuevas estrategias. De acuerdo a lo revisado, el análisis de polaridad de sentimientos es abordado con tres enfoques y para todos los casos se debe contar con un conjunto de palabras (diccionario) o mensajes (conjunto etiquetado) que sirvan como instrumento para determinar la polaridad de nuevos mensajes.

En esta tesis, para la identificación de mensajes prioritarios, conforme a lo comentado previamente, se emplea análisis de polaridad de sentimientos con el enfoque de clasificación usando aprendizaje supervisado, para tal efecto se hace uso de una base de mensajes etiquetados por la Sociedad Española para el Procesamiento de Lenguaje Natural (SEPLN) (24), una base propia de mensajes similares a los que analiza el DAI etiquetada por 5 voluntarios y la combinación de

ambas para contar con un conjunto de entrenamiento más grande; se estudia también el desempeño de los clasificadores en distintas configuraciones de las bases de mensajes y con dos tipos de representación de documentos TF (Frecuencia de Término) y TF-IDF (Frecuencia Inversa de Documento). Como parte de la estrategia de identificación de mensajes prioritarios se emplea además un enfoque basado en frecuencia de palabras de un diccionario creado a partir de un conjunto de mensajes prioritarios identificado previamente por el DAI y que permite realizar una discriminación de los mensajes que deben ser revisados.

A continuación se describen los conceptos técnicos necesarios para realizar análisis de polaridad de sentimientos conforme a lo expuesto anteriormente.

2.3 Minería de opinión y análisis de polaridad de sentimientos

La minería de opinión se encarga del estudio de los elementos de la opinión, siendo ésta un sentimiento, actitud, emoción o valoración respecto a una entidad (producto, servicio, persona, evento, organización o tema) o un aspecto de la entidad y que es emitida por un titular de opinión (3). Uno de los elementos del estudio de opinión es el análisis de sentimientos el cual incluye entre sus aspectos a la polaridad, la cual tiene por objetivo identificar la orientación semántica de la opinión como positiva, negativa o neutral.

El análisis de sentimientos en textos se puede abordar en tres niveles y se usan principalmente tres técnicas para su clasificación (14).

Los niveles en los que se realiza análisis de texto son los siguientes:

1. Sentencia. Para un documento dado el objetivo es identificar la polaridad de cada sentencia, además de identificar si el texto es objetivo o subjetivo, siendo éste al que se puede realizar análisis de sentimiento.

2. Documento. Se considera cualquier documento como una unidad de análisis para clasificarlo como positivo, negativo o neutral.

3. Atributo. Se identifican características particulares del contenido del texto que permiten identificar en un texto el objeto de la opinión y su polaridad, incluso se en una misma sentencia hay dos objetos de opinión con su propia polaridad (26).

Las tres técnicas principales para la clasificación de sentimientos son las siguientes:

1. Lexicón. Tiene dos enfoques, Diccionario y Corpus; el primero consiste en un conjunto de palabras etiquetadas y valuadas de acuerdo a la polaridad y fuerza del sentimiento que representan. Por su parte el corpus considera patrones sintácticos y semánticos en grandes conjuntos de palabras que, junto con una lista inicial de palabras de opinión, permiten relacionarlos con un sentimiento o polaridad en otros conjuntos de palabras (3).

2. Aprendizaje computacional. Se emplean algoritmos de clasificación supervisada y no supervisada. A partir de un conjunto de textos o documentos iniciales se entrena un clasificador para estimar la polaridad de un nuevo texto.

3. Método Híbrido. Ocupa la combinación de lexicones con aprendizaje computacional.

2.4 Clasificación supervisada de texto

La clasificación automática de texto tiene dos enfoques (27): clasificación supervisada y no supervisada. La primera consiste en entrenar un modelo de aprendizaje computacional a partir de un conjunto de documentos etiquetados para cada una de las clases, mientras la segunda se basa en características predefinidas a partir de recursos externos (26) como diccionarios de palabras valuados y/o en relaciones lingüísticas entre las palabras previamente definidas.

Particularmente los algoritmos de clasificación basados en aprendizaje supervisado permiten estimar a qué clase pertenece una entidad en función de sus características y de un clasificador entrenado con ejemplos de entidades etiquetadas previamente mediante un sistema de referencia manual; para el caso particular del análisis de polaridad los algoritmos de clasificación supervisada permiten asociar un documento o un mensaje a una de tres clases: positivo, negativo o neutral (25).

A continuación se describen los algoritmos que son utilizados en el análisis de esta tesis: Naive Bayes, Máquinas de vectores de soporte, K vecinos más cercanos y árboles de decisión.

2.4.1 Naive Bayes

Este algoritmo es un clasificador probabilístico que está basado en el Teorema de Bayes, el cual establece que la probabilidad condicional individual de un conjunto de eventos mutuamente excluyentes y exhaustivos A={A1, A2, ..., An} debidos a un evento B se puede calcular si se conocen la probabilidad de ocurrencia de B, la probabilidad de ocurrencia de B debido a Ai y la probabilidad de Ai, esto es:

$$P(B) = \frac{P(A_i) * P(A_i)}{P(B)}$$

Particularmente para su aplicación en la clasificación, los eventos Ai corresponden a cada una de las clases o etiquetas del fenómeno que se estudia y las probabilidades de los eventos Ai, B y de B debido a Ai se calculan mediante el enfoque frecuentista de probabilidad a partir de las características de conjunto de entrenamiento.

2.4.2 Máquinas de soporte vectorial

Este algoritmo fue desarrollado en los laboratorios de AT&T en los años 80; consiste en representar cada ejemplo del conjunto de entrenamiento como un punto en un espacio multidimensional, entonces todos los puntos (ejemplos) se separan en sus clases, alejados lo más posibles por hiperplanos denominados vectores de soporte. Los vectores de soporte establecen las fronteras entre clases, de esta manera es posible asignar un nuevo ejemplo a la clase que le corresponde.

Cuando los vectores de soporte no logran establecer las fronteras usando rectas, planos o hiperplanos es posible usar alguna otra función que sí lo permita y sea más flexible en la clasificación, tal como polinomios de grado n, funciones gaussianas o sigmoides, entre otras, esto tiene además la ventaja de permitir cierto grado de error en la exactitud, evitando así el sobreajuste.

2.4.3 K vecinos más cercanos

Este algoritmo calcula la similitud entre las características de los ejemplos del conjunto de entrenamiento y agrupa en las distintas clases aquellos que tengan los índices de similitud más cercanos entre sí. De esta manera, cuando un nuevo

ejemplo requiere ser clasificado, se asocia a la clase en la cual los k ejemplos fueron mayoría al obtener un máximo índice de similitud.

2.4.4 Árboles de decisión

Los árboles de decisión son algoritmos basados en grafos dirigidos que parten de un nodo raíz y se extienden mediante ramas hasta representar un conjunto que describe a los ejemplos; las conexiones entre nodos se establecen mediante reglas o condiciones que se extraen de las características del conjunto de ejemplos de entrenamiento. Cuando un nuevo ejemplo es evaluado con un árbol de decisión, sus características se van comparando con las condiciones de cada nodo y va recorriendo las ramas hasta encontrar una clase que lo represente.

2.5 Relevancia y vectorización de texto

Los algoritmos de aprendizaje descritos en 2.4 emplean como datos de entrada para el procesamiento un conjunto de vectores estandarizados en contenido y forma, razón por la cual es necesario realizar una transformación del mensaje o documento en un vector que represente el texto contenido en cada uno de éstos.

Los métodos más usados y más sencillos para vectorización de documentos es su transformación en valores ponderados en función del número de veces que se usa cada término de un conjunto de palabras predeterminadas.

El conjunto de palabras o bolsa de palabras (28) corresponde a un diccionario creado a partir de todos los documentos que se analizan. Para cada mensaje o documento se cuentan las palabras que contiene y se asigna un valor a una posición en el vector. En principio el vector tiene el mismo tamaño que el total de palabras del diccionario.

Para determinar el valor de cada término del mensaje y asignarlo al vector se emplean métodos aritméticos, siendo los más usados la ponderación binaria (se asigna 1 ó 0 a cada posición del vector si existe o no en cada mensaje la palabra del diccionario), la frecuencia absoluta (se asigna a cada posición del vector el número de veces que cada palabra del diccionario se repite), la frecuencia relativa denominada comúnmente como TF (que es la frecuencia absoluta estandarizada conforme al total de palabras en el mensaje) y el método

TF-IDF que incluye lo que se conoce como Frecuencia Inversa de Documento (Inverse Document Frequency).

A diferencia de los 3 primeros métodos de ponderación, que sólo consideran la importancia de cada palabra en su propio documento, TF-IDF considera la importancia de las palabras de cada documento respecto al conjunto de todos los documentos que se analizan. La expresión para determinar el valor de TF-IDF es la siguiente:

$$TFIDF = TF * log(\frac{D}{1 + nDt})$$

Donde:

TF es la frecuencia relativa de término.

D es el total de documentos o mensajes que conforman el corpus bajo estudio.

nDt es el total de documentos o mensajes donde aparece el término t.

2.6 Medidas de evaluación de desempeño

Para medir el desempeño de los modelos de clasificación supervisada se emplean distintas métricas en función de cuatro casos que se pueden presentar en la predicción del clasificador, mismos que son descritos por la Matriz de Contingencia (Cuadro 1).

Predicción/Clase	Clase A	Clase B
Positiva para A	**Verdadero Positivo (TP)** Ej. Persona enferma con resultado positivo para la enfermedad	**Falso Positivo (FP)** Ej. Persona sana con resultado positivo para una enfermedad.
Negativa para A	**Falso Negativo (FN)** Ej. Persona enferma con resultado negativo para la enfermedad.	**Verdadero Negativo (TN)** Ej. Persona sana con resultado negativo de enfermedad.

Cuadro 2. Matriz de Contingencia. Casos posibles de la predicción de resultados. Fuente: elaboración propia basada en (28).

Las medidas de evaluación son las siguientes:

Exactitud. Es la relación entre el número de predicciones correctas y el total de predicciones realizadas por el clasificador. Esta medida de precisión se recomienda cuando las clases y el corpus se encuentra balanceado.

$$\boldsymbol{Exactitud} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{FP} + \boldsymbol{TN} + \boldsymbol{FN}}$$

Precisión. Es una medida de evaluación a nivel de clase; se calcula como la relación entre el número de predicciones correctas de la clase A respecto al total de predicciones de la misma clase.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

Exhaustividad. También se conoce como Recall, es la relación entre el número de predicciones correctas de la clase A respecto al total de documentos de la clase A y permite estimar en qué proporción el modelo es capaz de representar correctamente las clases.

$$\boldsymbol{Exhaustividad} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

F-Score. Es una combinación de las medidas de Precisión y Exhaustividad, ponderada por un parámetro k que permite dotar de mayor importancia a una de las dos medidas, aunque regularmente k = 1 y a la medida de evaluación se le denomina entonces F1-Score.

$$\boldsymbol{F_k Score} = (\mathbf{1} + \boldsymbol{k^2}) * \frac{\boldsymbol{Precision} * \boldsymbol{Exhaustividad}}{(\boldsymbol{k^2} * \boldsymbol{Precision}) + \boldsymbol{Exhaustividad}}$$

El desarrollo de esta tesis se aborda en tres enfoques para identificar mensajes prioritarios del DAI:

- aplicando técnicas de clasificación supervisada para análisis de polaridad, empleando una base de mensajes etiquetados a partir de valoración humana como conjunto de entrenamiento;

- usando un diccionario de palabras significativas, creado a partir de un conjunto de mensajes identificados previamente por el DAI por su contenido de interés al desempeño institucional.
- combinando los dos enfoques anteriores, es decir a partir de la clasificación de polaridad, identificar en la clase negativa mensajes prioritarios basados en el diccionario de palabras significativas.

Los mensajes prioritarios, conforme a lo planteado en 1.1 Antecedentes y 1.2 Problemática, son aquéllos que por su contenido tienen el potencial de afectar de forma mediática o funcional la eficacia de la institución.

La identificación de mensajes que pueden afectar mediáticamente se aborda con técnicas de análisis de polaridad que, como se revisó en la sección en mientras que los mensajes que pueden afectar las funciones de la institución se identifican mediante un diccionario de palabras significativas. La combinación de ambos enfoques permitirá identificar si las afectaciones mediáticas pueden ser amenazas a la eficacia institucional.

En el siguiente capítulo se describe la metodología empleada así como su aplicación en el análisis de mensajes prioritarios.

Capítulo 3

Metodología y Análisis

Capítulo 3. Metodología y Análisis

La identificación de mensajes prioritarios que realiza el DAI depende de dos aspectos que se juzgan de manera independiente por un analista mediante lectura directa durante su jornada laboral: la carga emocional que puede afectar la imagen y aceptación social de la institución (o de quien la institución determine como objeto o sujeto de análisis) y el contenido de palabras significativas relativas a eventos o hechos que pueden afectar el desempeño institucional.

Resulta necesario que cada mensaje de Twitter recuperado por el DAi sea clasificado conforme a su carga emocional, tal que pueda identificarse si la mención para la institución o el sujeto de estudio es al menos positiva, negativa o neutral; así mismo se requiere identificar mensajes con palabras significativas en el ámbito de competencia de la institución, tal que podrían suponer un riesgo a sus funciones o ser una fuente de información para la toma de decisiones.

Considerando lo anterior se abordan en este trabajo dos enfoques independientes y un enfoque combinado para analizar y evaluar algoritmos para la clasificación e identificación de mensajes prioritarios: clasificación automática de polaridad emocional, frecuencia de palabras significativas conforme a un diccionario (de interés institucional) y frecuencia de palabras significativas en la clase negativa de la polaridad emocional, que se agrega como una aportación de esta tesis para el DAI bajo la consideración de que las menciones negativas no solo podrán afectar la imagen o percepción del sujeto de estudio, sino también incluir palabras significativas de interés institucional que podrían afectar su desempeño (este último se aborda como un resultado de la tesis en la sección 5 de este documento).

A continuación se describen la metodología empleada para el análisis y generación de resultados; en los títulos subsecuentes se explica el desarrollo del análisis para cada una de los dos enfoques: clasificación de polaridad e identificación de mensajes prioritarios.

3.1 Metodología

Para el desarrollo del presente trabajo se emplea como metodología de análisis de mensajes de Twitter el enfoque propuesto por Yoon, Elhadad y Bakken (29),

mismo que ha sido usado en otras investigaciones (9, 10, 20, 21), el cual consta de los siguientes 5 pasos:

1. Análisis conceptual. En este paso se define el objetivo del análisis y se identifican palabras clave que lo representen para recopilar un conjunto inicial de mensajes de Twitter.

2. Recopilación de datos. En este paso se recopilan y revisan mensajes existentes, en caso de que se requieran más textos deberán implementarse instrumentos, métodos y herramientas para su recuperación.

3. Preparación de datos. Los mensajes recuperados deben acondicionarse y transformarse para su uso confiable, es decir que los mensajes no contengan fuentes de ruido o distorsión y que su formato sea útil y válido para el análisis posterior. Se emplean técnicas para vectorización y valorización de la relevancia del texto.

4. Análisis de datos. Se aplican las técnicas de clasificación y análisis para desarrollar y entrenar modelos que sean capaces de clasificar mensajes nuevos.

5. Interpretación de datos. El modelo debe ser evaluado para determinar su exactitud. Los resultados se presentan gráficamente para su comprensión e interpretación. Una vez que el proceso de análisis ha concluido, a partir de éste es posible plantear nuevos objetos de estudio.

La figura 1 muestra el enfoque de análisis para mensajes de Twitter descrito anteriormente.

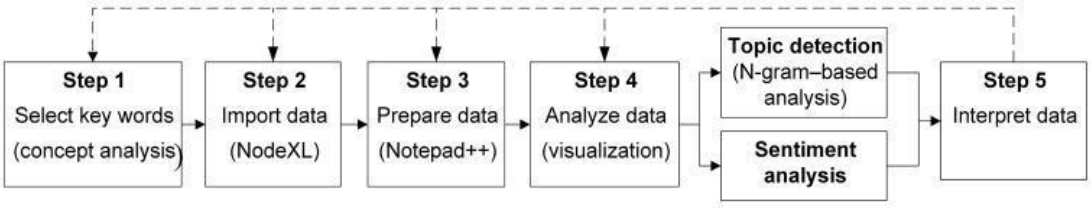


Figura 1. Método de análisis de mensajes de Twitter.
Fuente: Yoon, Elhadad y Bakken (28).

A continuación se describe la aplicación del método antes descrito en la solución que se propone en este trabajo, la cual consiste en realizar clasificación de polaridad de mensajes de Twitter y en seleccionar mensajes en función de sus palabras significativas contenidas en un diccionario elaborado para tal fin, lo

anterior para identificar mensajes prioritarios que puedan afectar a la DAI de forma mediática u operacional.

3.2 Análisis conceptual

En el ámbito de este trabajo y de la problemática planteada en la sección 1.2, se realiza el análisis con el objetivo de identificar mensajes prioritarios relativos a instituciones, organizaciones, personas, eventos, coyunturas, noticias o hechos, desde dos enfoques:

1. La polaridad de un mensaje o conjunto de mensajes, valorada como positivo, negativo o neutral.

2. La importancia de un mensaje por su contenido de palabras significativas en el contexto institucional.

Para ambos casos se realiza la recuperación de mensajes de Twitter relacionados con temáticas de diversa índole en idioma español, los cuales pueden incluir carga emocional y contenido de interés para la institución.

3.3 Recopilación de datos

Con el objetivo de desarrollar los modelos para analizar los dos enfoques de identificación de mensajes prioritarios se emplean 2 bases de datos etiquetadas para entrenar algoritmos de clasificación de la polaridad y una base de datos de mensajes prioritarios para elaborar un diccionario de palabras significativas.

Respecto a las dos bases de datos etiquetadas para realizar análisis de la polaridad, una fue solicitada a la Sociedad Española para el Procesamiento de Lenguaje Natural (SEPLN) y la otra fue integrada como parte de este trabajo mediante twits obtenidos usando la API de Twitter[3], ambas bases de datos se describen en las secciones 3.3.1 y 3.3.2.

La base de datos de mensajes prioritarios fue proporcionada por la DAI y se describe en la sección 3.3.3.

[3] Twitter ofrece 4 tipos de cuentas para desarrolladores, la cuenta estándar es libre y permite recuperar alrededor de 3 mil mensajes en los últimos 6 a 9 días a partir de una búsqueda particular. La Api de Twitter devuelve además otros datos como el usuario que lo generó, la ubicación geográfica, número de seguidores y cuentas seguidas, URL de imágenes embebidas en el mensaje, fecha y hora de publicación, identificador de usuario, entre otros. Fuente: https://developer.twitter.com/

3.3.1 Base de datos de la SEPLN

Desde 2012 la Sociedad Española para el Procesamiento de Lenguaje Natural organiza periódicamente el Taller de Análisis de Sentimientos. Uno de los trabajos que en este taller se realiza por un grupo de especialistas es la integración de conjuntos de mensajes etiquetados con algún grado de emoción, así se tienen enunciados calificados como positivos, negativos, neutros o ninguno de los anteriores.

Para obtener las bases de datos es necesario solicitarlas mediante correo electrónico a la SEPLN, quien proporciona una liga de acceso al repositorio de mensajes etiquetados.

En el desarrollo del presente trabajo se empleó la base de datos de Español Internacional 2018 (Inter-TASS corpus 2018), la cual consta de 2 conjuntos de 506 y 1008 mensajes de Twitter etiquetados como Positivo, Negativo Neutro o Ninguno (30).

Con fines prácticos ambos conjuntos se integran en un único grupo de 1514 mensajes; todos los mensajes etiquetados como Ninguno se consideraron como Neutros. La distribución porcentual de cada grado de sentimiento es: Negativo, 42%; Positivo, 31%; Neutral 27%.

Adicionalmente, a partir del mismo conjunto de 1514 mensajes, se creó uno nuevo solo para mensajes positivos y negativos, de esta manera se constituye una base de 1111 mensajes con la siguiente distribución: Positivo, 43%; Negativo 57%.

3.3.2 Base de datos de integración propia

Con el objetivo de contar con una base de mensajes de interés, similares a los que regularmente recupera el DAI como parte de su proceso de análisis (tal como se comenta en la sección 1.2), se integró una base de mensajes de Twitter en idioma español sobre distintos temas, lugares, personas, organizaciones y eventos.

Usando la API de Twitter con una cuenta de desarrollador estándar se recuperaron, entre los meses de Julio y Diciembre de 2019, mensajes de los siguientes tópicos, los cuales en dicho periodo correspondieron a temas de significativa importancia en el contexto nacional y que por su diversidad temática

contienen estilos de discurso distintos que se espera aprenda el algoritmo de clasificación para emplearse en cualquier tema:

- Temas: #cobardematoncito, #byebyebeetle, #PrimerInformeGobMx
- Lugares: #Puebla
- Personas: #BarbosaGobernador, @CarlosUrzuaSHCP
- Organizaciones:@semar_mx, sedena_mx, #300lideres
- Eventos: #TourdeFrancia, #JuegosPanamericanos2019

Con los mensajes recuperados se integró un conjunto de 1911 twits originales, es decir que fueron escritos por su autor, en ningún caso se usaron mensajes replicados por usuarios (retwits) y en algunos casos los mensajes eran respuesta a otro twit.

Considerando que los 1911 mensajes corresponden en su mayoría a temas de interés regional o nacional y que pudieran incluir acepciones propias del idioma español se optó por identificar su polaridad mediante discernimiento humano.

Para etiquetar la polaridad, el conjunto de mensajes fue repartido aleatoriamente a 5 voluntarios quienes se encargaron de asignar la etiqueta 1, -1 ó 0, en función de la carga emocional, esto es positivo, negativo o neutral, respectivamente. Los 5 voluntarios que apoyaron en esta tarea son estudiantes mexicanos de ciencias e ingeniería. Este grupo de evaluadores fue instruido previamente para reducir la posibilidad de cualquier sesgo debido a la afinidad o empatía de opinión con el mensaje evaluado.

Para determinar la polaridad o carga emocional de cada mensaje, los voluntarios identificaron si las palabras usadas y el mensaje escrito manifestaban una carga positiva o negativa, procurando evitar cualquier juicio de valor personal o preferencia política, sólo se infirió si las expresiones usadas en los twits transmitían una emoción positiva o negativa. Los mensajes que tuvieron un objetivo informativo sin algún juicio emocional, se consideran neutros. El procedimiento empleado para realizar este etiquetado con discernimiento humano se realizó en 5 jornadas de 8 horas conforme al siguiente procedimiento:

1. Se generaron aleatoriamente 4 grupos de 382 mensajes y 1 grupo de 383 mensajes.

2. Cada grupo de mensajes fue asignado a un evaluador quien se encargaba de leer cada mensaje y determinar la orientación semántica del texto como positiva, negativa o neutral. Los mensajes que contenían palabras ofensivas o agresivas respecto a un sujeto u objeto en el contexto del mensaje se etiquetaron como negativos asignando el valor “-1”. El uso de palabras ligadas al bienestar, buenos deseos, éxito u optimismo fueron consideradas positivas y se etiquetaron con “1”. Los mensajes con una connotación informativa, sin una clara orientación semántica se consideraron neutros y se etiquetaron con “0”.
3. Se realizaron 4 rondas de etiquetado, cada uno de los 5 voluntarios recibió y etiquetó 4 grupos de mensajes sin conocer las etiquetas asignadas previamente por sus compañeros.
4. Al finalizar las 4 rondas se integraron todas las etiquetas asignadas por los voluntarios; cada uno de los mensajes de los 5 grupos contó con 4 etiquetas asignadas por los voluntarios; cada mensaje fue revisado y se le asignó como etiqueta final la que presentaba mayoría.
5. Los mensajes que presentaron empate fueron enviados al quinto voluntario que inicialmente no lo etiquetó y su valuación se consideró definitiva.

La distribución de la carga emocional obtenida para este conjunto de mensajes es la siguiente: Negativo, 32%; Positivo, 22%; Neutral 46%.

Adicionalmente se dispuso de la misma base datos sin considerar el grado de sentimiento Neutral, de esta manera se constituye una base de 1022 mensajes con la siguiente distribución: Positivo, 40%; Negativo 60%.

3.3.3 Base de datos de interés institucional

Con el objetivo de identificar mensajes de contenido prioritario para los intereses del DAI, éste proporcionó una pequeña base de datos de 256 mensajes que fueron catalogados por sus analistas en Septiembre de 2018 como casos de seguimiento debido a su contenido.

Como se explica más adelante, esta base de datos fue empleada para construir un diccionario de palabras significativas para su uso en la identificación

de mensajes prioritarios por su potencial de afectar la eficacia de las funciones del DAI y la institución a la que pertenece.

3.3.4 Base de datos de prueba

Para probar los modelos de clasificación de la polaridad se emplean dos bases de mensajes coyunturales que fueron recuperados usando la API de Twitter con los hashtag #PrensaProstituida (14,649 twits) y #CobardeMatoncito (65,406 twits) ambos casos al considerarse temas con alta carga de contenido emocional, que en su momentos fueron tópico de tendencia y que tenían el potencial de afectar mediáticamente al gobierno mexicano.

Por su parte, para probar el modelo de identificación de mensajes prioritarios por diccionario se empleó una base de 3717 mensajes de Twitter, recuperados entre el 1 y el 7 de enero de 2020 que mencionan a la cuenta institucional @SEMAR_mx misma que pertenece al gobierno mexicano y que realiza tareas similares a la institución a la que pertenece el DAI, motivo por el cual se emplea para su análisis en este trabajo.

En el Capítulo 4 Resultados, de este documento, se explica el uso que se dio a las 3 bases de datos de prueba comentadas en este apartado.

3.4 Preparación de datos

Para generar los modelos de clasificación se debe realizar la limpieza y estandarización del texto, lo cual facilita su análisis posterior. En este apartado se explica el procesamiento que se realiza al texto para su estandarización y transformación vectorial.

3.4.1 Procesamiento de texto

A continuación se describe cada tarea de limpieza y estandarización que se realiza a cada mensaje de Twitter así como el código utilizado en lenguaje Python, se parte de asignar un archivo de mensajes a un arreglo denominado como *"datext"* en este análisis.

- Eliminación de acentos de las palabras, empleando el siguiente código:

```
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
```

```
    l = l.lower()
    l = l.translate(trans)
dat.append(l)
```

- Generación de tokens, es decir separación de enunciados en palabras, usando la librería *word_tokenize* de NLTK[4] (31) en el siguiente código:

```
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
tokens.append(toks)
```

- Eliminación de *stopwords* o palabras vacías, usando la lista para idioma Español de NLTK:

```
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in
                      stopwords.words('spanish')]
    fw.append(filtered_words)
    filtered_words.clear
```

- Reducción de palabras a su raíz (stemming), para lo cual se utiliza el algoritmo de Porter disponible en *NLTK*:

```
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
```

Con el procesamiento de texto antes descrito, un mensaje de Twitter es reducido a un conjunto menor de palabras que en el mensaje original, pero con mayor significancia para la construcción de modelos de clasificación.

3.4.2 Representación vectorial del texto

Una vez que se cuenta con un arreglo de mensajes tokenizados, si se desea realizar clasificación supervisada con algoritmos de aprendizaje computacional, es

[4] Natural Lenguage Tool Kit (NLTK) es un conjunto de librerías para el procesamiento y análisis de lenguaje natural simbólico.

necesario transformarlos en una representación numérica donde cada mensaje se describe con un vector numérico.

Para vectorizar los mensajes tokenizados se empleó el índice de frecuencia de término (TF) y el índice de frecuencia inversa de documento (TF-IDF), descritos en la sección 2.5 de este documento, los cuales están implementados en las funciones *CountVectorizer* y *TfidfVectorizer* de la librería *feature_extraction.text* de SciKit-Learn[5] (30).

El código empleado para la vectorización de un arreglo de tokens (denominado stem[] en este caso), usando índice TF, es el siguiente:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
features = vectorizer.fit_transform(stem).todense()
```

El código empleado para la vectorización de un arreglo de tokens usando TF-IDF es el siguiente:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
        features = vectorizer.fit_transform('arreglo_tokens').toarray()
```

3.5 Análisis de datos

El análisis de los datos se realizó con los dos enfoques asociados a la identificación de mensajes prioritarios que realiza el DAI:

- análisis de polaridad, para las 2 bases de datos TASS (2 y 3 clases) y las 2 bases de datos propias (2 y 3 clases).
- prioridad basada en diccionario de palabras significativas, para la base de datos de mensajes de interés proporcionada por el DAI.

Para el caso del análisis de polaridad de mensajes se revisaron modelos de clasificación basados en aprendizaje supervisado. Por su parte, para la identificación de mensajes prioritarios se empleó clasificación no supervisada a

[5]Scikit-Learn es una librería de análisis predictivo de datos basado en aprendizaje computacional

partir de la construcción de un diccionario. En ambos caso se realizó primero el procesamiento de texto para su limpieza y estandarización.

La construcción de los modelos analíticos se realizó en lenguaje Python sobre la plataforma Jupyter, utilizando librerías especializadas en estructuras de datos (Pandas[6]) (29), análisis de texto (NLTK) y aprendizaje computacional (SciKit-Learn).

A continuación se describe el análisis realizado para cada uno de los dos enfoques requeridos por el DAI.

3.5.1 Modelos de análisis de polaridad basados en clasificación supervisada

Para generar modelos de análisis de polaridad se emplearon las librerías de clasificación supervisada de *SciKit-Learn,* se implementaron cuatro algoritmos de clasificación: Máquinas de Soporte Vectorial (SVC), Naive Bayes (Gaussian NB), Centroide más cercano (NearestCentroids) y Árboles de decisión (DecisionTree).

Cada uno de los cuatros clasificadores se entrenó y evaluó para los siguientes casos:

- BD TASS vectorizada con TF para 3 clases (Positiva, Negativa y Neutral).
- BD TASS vectorizada con TF-IDF para 3 clases (Positiva, Negativa y Neutral).
- BD propia vectorizada con TF para 3 clases (Positiva, Negativa y Neutral).
- BD propia vectorizada con TF-IDF para 3 clases (Positiva, Negativa y Neutral).
- BD TASS vectorizada con TF para 2 clases (Positiva y Negativa).
- BD TASS vectorizada con TF-IDF para 2 clases (Positiva y Negativa).
- BD propia vectorizada con TF para 2 clases (Positiva y Negativa).
- BD propia vectorizada con TF-IDF para 2 clases (Positiva y Negativa).
- BD TASS + BD propia vectorizada con TF-IDF para 3 clases (Positiva, Negativa y Neutral).
- BD TASS + BD propia vectorizada con TF-IDF para 2 clases (Positiva y Negativa).

[6] Python Data Analysis Library (Pandas) es una librería que facilita el análisis y visualización de estructuras de datos.

- BD TASS vectorizada con TF-IDF para 3 clases, usada como conjunto de entrenamiento para predecir BD propia como conjunto de prueba.
- BD TASS vectorizada con TF-IDF para 2 clases, usada como conjunto de entrenamiento para predecir BD propia como conjunto de prueba.
- BD propia vectorizada con TF-IDF para 3 clases, usada como conjunto de entrenamiento para predecir BD TASS como conjunto de prueba.
- BD propia vectorizada con TF-IDF para 2 clases, usada como conjunto de entrenamiento para predecir BD TASS como conjunto de prueba.

Para evaluar los clasificadores de los casos descritos anteriormente (excepto los últimos cuatro), se empleó validación cruzada de 10 lotes, la cual fue implementada con la función *cross_val_score* de la librería *model_selection* de *SciKit-Learn.* El código implementado para entrenar y evaluar cada clasificador es el siguiente:

```
clf = 'clasificador'()
scores = cross_val_score(clf, 'mensajes_vectorizados', 'vector_etiq_clases',
                         cv=10, scoring='precision_micro')
```

El clasificador (*'clasificador'*) se implementa como *SVC()* para máquinas de soporte vectorial, *GaussianNB()* para *Naive Bayes*, *NearestCentroid()* para centroide más cercano y *tree.DecisionTreeClassifier()* para árboles de decisión. Las medidas de evaluación empleadas (*scoring*) se implementan como *'accuracy'* para exactitud, *'precision_micro'* para precisión multiclase, *'recall_micro'* para exhaustividad multiclase y *'F1_micro'* para F1 multiclase, mismas que están disponibles como funciones de la librería *metrics* de *SciKit-Learn.*

En la sección 4.5 Interpretación de datos, se muestran los resultados de cada uno de los casos de clasificación aquí mencionados.

3.5.2 Modelo de clasificación de prioridad basado en diccionario

Para identificar mensajes considerados como prioritarios, a partir de ejemplos identificados por la DAI, se generó un diccionario de palabras

significativas y se propuso un modelo basado en la frecuencia de éstas para ordenar los mensajes, según se explica a continuación.

3.5.2.1 Generación de diccionario de términos significativos

A partir de una pequeña base de mensajes de interés para la Institución (sección 3.3.3), considerados como prioritarios para su seguimiento (debido a su contenido coyuntural), se recuperaron las palabras significativas y se integraron en un diccionario, siendo éstas principalmente verbos, sustantivos y nombres.

Inicialmente se realizó el procesamiento de texto descrito en la sección 3.4.1 de este documento para generar un listado de tokens, reduciendo los mensajes a un conjunto de vectores de palabras individuales, en minúsculas, sin signos de puntuación y sin considerar palabras vacías (stop words).

Posteriormente se usó la función *CountVectorizer* de la librería *feature_extraction.text* de *SciKit-Learn* para generar un diccionario a partir de un arreglo de tokens (stems para este caso), según se muestra en el siguiente código:

```
from sklearn.feature_extraction.text import CountVectorizer
features = vectorizer.fit_transform(stems)
diccionario = vectorizer.vocabulary_
```

Finalmente se realizó una inspección visual del contenido del diccionario para eliminar términos no deseados, por ejemplo sustantivos compuestos o enunciados cortos de dos o más palabras que no fueron filtrados en el procesamiento de texto.

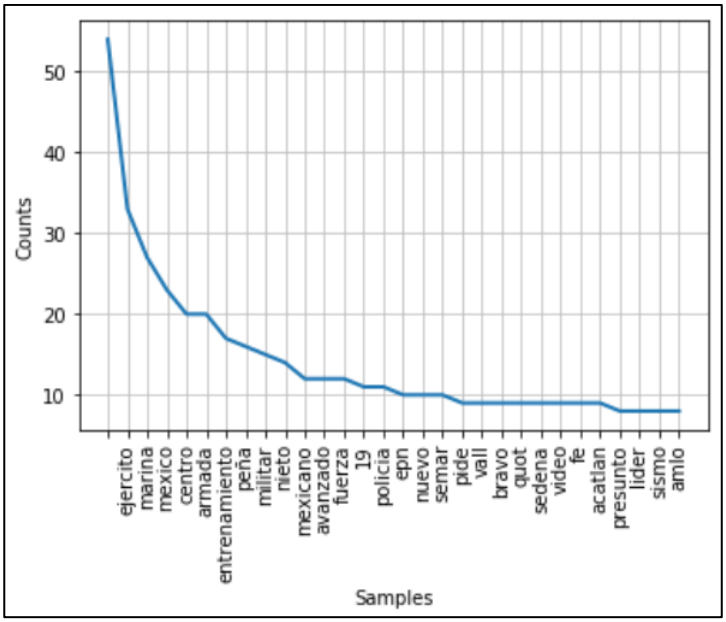


Gráfico 1. Gráfico de Zipf de las 30 palabras más frecuentes en la base de mensajes. Fuente: elaboración propia

El diccionario integrado contiene 555 palabras. La figura 5 muestra el Gráfico de Zipf, que corresponde a las 30 palabras más frecuentes en el conjunto de mensajes bajo análisis.

3.5.2.2 Modelo de prioridad basado en frecuencia de término

Para identificar mensajes prioritarios se propone un modelo basado en frecuencia de término de diccionario, esto es, para cada mensaje nuevo se realiza la identificación y conteo de las palabras del diccionario que contenga, entonces los mensajes se priorizan en función de la frecuencia total de palabras del diccionario que contengan.

El código empleado para identificar y contar palabras es el siguiente:

```
etik =[]
for lin in stem:
    for w in lin:
        if w in diccionario: i=i+1
    etik.append(i)
    i=0
```

Donde, *etik* es el arreglo que guarda la frecuencia de términos de diccionario de cada mensaje y *stem* contiene los conjuntos de tokens de cada mensaje que integra el conjunto de mensajes de prueba.

Para una base de datos de mensajes nuevos, en un periodo de tiempo dado, es posible identificar aquellos con mayor cantidad de palabras significativas y priorizarlos para su revisión por parte de un analista.

3.6 Interpretación de datos

A continuación se muestran los resultados de las métricas de evaluación de los distintos casos de estudio que se revisaron para los clasificadores de polaridad emocional así como el resultado de la clasificación por prioridad de diccionario usando la misma base de datos que lo generó.

3.6.1 Evaluación de modelos de clasificación de polaridad emocional

A continuación se presentan los resultados de las métricas de evaluación para los clasificadores supervisados, usando validación cruzada y validación por conjunto de entrenamiento, descritos en la sección 2.6.

- BD TASS vectorizada con TF y TF-IDF para 3 clases (Positiva, Negativa y Neutral).

10 Cross Validation Base de Datos TASS. Clases: Positivo, Negativo, Neutral								
	Vectorización TF				Vectorización TF - IDF			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.51	0.51	0.51	0.51	0.53	0.53	0.53	0.53
NaiveBayes	0.41	0.41	0.41	0.41	0.4	0.4	0.4	0.4
NearCentroids	0.51	0.51	0.51	0.51	0.52	0.52	0.52	0.52
DecisionTree	0.48	0.48	0.48	0.48	0.48	0.47	0.46	0.46

Cuadro 3. Evaluación de clasificadores para 3 clases de la BD TASS.
Fuente: elaboración propia

- BD propia vectorizada con TF y TF-IDF para 3 clases (Positiva, Negativa y Neutral).

10 Cross Validation Base de Datos Propia. Clases: Positivo, Negativo, Neutral								
	Vectorización TF				Vectorización TF - IDF			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.49	0.49	0.49	0.49	0.52	0.52	0.52	0.52
NaiveBayes	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46
NearCentroids	0.41	0.41	0.41	0.41	0.48	0.48	0.48	0.48
DecisionTree	0.49	0.49	0.49	0.49	0.47	0.48	0.47	0.49

Cuadro 4. Evaluación de clasificadores para 3 clases de la BD Propia.
Fuente: elaboración propia

- BD TASS vectorizada con TF y TF-IDF para 2 clases (Positiva y Negativa).

10 Cross Validation Base de Datos TASS. Clases: Positivo, Negativo								
	Vectorización TF				Vectorización TF - IDF			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.71	0.71	0.71	0.71	0.73	0.73	0.73	0.73
NaiveBayes	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.61
NearCentroids	0.69	0.69	0.69	0.69	0.72	0.72	0.72	0.72
DecisionTree	0.68	0.67	0.68	0.66	0.64	0.64	0.64	0.64

Cuadro 5. Evaluación de clasificadores para 2 clases de la BD TASS.
Fuente: elaboración propia

- BD propia vectorizada con TF y TF-IDF para 2 clases (Positiva y Negativa).

10 Cross Validation Base de Datos Propia. Clases: Positivo, Negativo								
	Vectorización TF				Vectorización TF - IDF			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.75	0.75	0.75	0.75	0.78	0.78	0.78	0.78
NaiveBayes	0.74	0.74	0.74	0.74	0.73	0.73	0.73	0.73
NearCentroids	0.69	0.69	0.69	0.69	0.73	0.73	0.73	0.73
DecisionTree	0.72	0.72	0.74	0.74	0.69	0.69	0.7	0.68

Cuadro 6. Evaluación de clasificadores para 2 clases de la BD Propia.
Fuente: elaboración propia

Como se aprecia en las Cuadros 3 a 6, el clasificador SVM (Máquinas de vectores de soporte) con vectorizado TF-IDF, es el que presenta las mejores métricas de evaluación respecto al resto de clasificadores.

A continuación se muestran los resultados de los clasificadores usando de manera conjunta las Base de Datos TASS y la Base de Datos Propia; se propone concatenar ambas bases para incrementar la base de conocimiento en el entrenamiento y prueba de los clasificadores. Para estos casos de estudio sólo se empleó vectorizado TF-IDF ya que en los ejemplos antes descritos presenta mejor desempeño en la clasificación; los clasificadores se entrenaron y probaron para tres clases (positivo, negativo y neutral) y para dos clases (positivo y negativo).

- BD TASS + BD propia vectorizada con TF-IDF para 3 clases (Positiva, Negativa y Neutral) y para 2 clases (Positiva y Negativa).

10 Cross Validation BD Propia + BD TASS Vectorizado TF-IDF								
	Tres clases: Positivo, Negativo, Neutral				Dos clases: Positivo y Negativo			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75
NaiveBayes	0.43	0.43	0.43	0.43	0.65	0.65	0.65	0.65
NearCentroids	0.5	0.5	0.5	0.5	0.73	0.73	0.73	0.73
DecisionTree	0.45	0.44	0.45	0.44	0.66	0.67	0.66	0.66

Cuadro 7. Evaluación de clasificadores para 2 y 3 clases BD TASS + BD Propia.
Fuente: elaboración propia

Como puede apreciarse en el Cuadro 7, no se percibe mejora al concatenar ambas bases de datos, de acuerdo a los resultados de las métricas de evaluación, la capacidad de clasificación es similar a los clasificadores generados a partir de las bases de datos individuales.

A continuación se presentan los resultados de los clasificadores entrenados con la base de datos TASS y probados con la base de datos propia, para los casos de 2 clases (Positivo y Negativos), así como para 3 clases (Positivo, Negativo y Neutral).

- BD TASS vectorizada con TF-IDF para 3 clases y 2 clases, usada como conjunto de entrenamiento para predecir BD Propia (conjunto de prueba).

Evaluación Conjunto de Entrenamiento: BD TASS, Conjunto de Prueba: BD Propia								
	Tres clases: Positivo, Negativo, Neutral				Dos clases: Positivo y Negativo			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.35	0.35	0.35	0.35	0.63	0.63	0.63	0.63
NaiveBayes	0.35	0.35	0.35	0.35	0.55	0.55	0.55	0.55
NearCentroids	0.39	0.39	0.39	0.39	0.64	0.6	0.64	0.64
DecisionTree	0.35	0.35	0.35	0.35	0.6	0.6	0.6	0.6

Cuadro 8. Evaluación de clasificadores de 2 y 3 clases, Entrenamiento con BD TASS, Prueba con BD Propia.
Fuente: elaboración propia

- BD Propia vectorizada con TF-IDF para 3 clases y 2 clases, usada como conjunto de entrenamiento para predecir BD TASS (conjunto de prueba).

Evaluación Conjunto de Entrenamiento: BD Propia, Conjunto de Prueba: BD TASS								
	Tres clases: Positivo, Negativo, Neutral				Dos clases: Positivo y Negativo			
Clasif \ Métrica	Exactitud	Precisión	Exhaustividad	F1	Exactitud	Precisión	Exhaustividad	F1
SVM	0.3	0.3	0.3	0.3	0.64	0.64	0.64	0.64
NaiveBayes	0.38	0.38	0.38	0.38	0.6	0.6	0.6	0.6
NearCentroids	0.3	0.3	0.3	0.3	0.63	0.63	0.63	0.63
DecisionTree	0.32	0.32	0.32	0.32	0.59	0.59	0.59	0.59

Cuadro 9. Evaluación de clasificadores de 2 y 3 clases, entrenamiento con BD TASS, prueba con BD Propia.
Fuente: elaboración propia

De los cuadros 8 y 9 puede observarse que el comportamiento de los clasificadores es muy parecido para los cuatro casos que se probaron, ninguna de las bases de datos ofrece una ventaja respecto a la otra, por lo que puede aprovecharse la concatenación de ambas para contar con más ejemplos útiles en la clasificación, además de que su concatenación presentó un buen desempeño en la validación cruzada (Cuadro 7).

3.6.2 Comportamiento del modelo de prioridad de diccionario

El modelo de prioridad de mensajes basado en diccionario se generó a partir de una pequeña base de mensajes de interés, usando esa misma base para probar el modelo.

Utilizando la función *tabulate* de la librería *FreqDist* de *NLTK.probability* se calcula la cantidad de mensajes que corresponden a cada una de las frecuencias de términos prioritarios identificadas en el corpus.

El cuadro 10 muestra la frecuencia de término basada en diccionario y la cantidad de mensajes del corpus correspondiente a cada frecuencia.

Mensajes clasificados por Frecuencia de Témino basada en diccionario										
Frecuencia de Términos	0	1	2	3	4	5	6	7	8	10
Cantidad de Mensajes	8	25	43	63	47	36	26	6	1	1

Cuadro 10. Mensajes del corpus para cada Frecuencia de término de diccionario.
Fuente: Elaboración propia.

Del Cuadro 10 se destaca que, a pesar que los 256 mensajes son considerados de interés coyuntural, 33 podrían descartarse por incluir menos de 2 palabras del diccionario; 153 mensajes se consideran de interés medio dado que contienen entre 2 y 4 palabras del diccionario, 68 pudieran considerase de interés

prioritario por contener entre 5 y 7 palabras, y 2 mensajes son de alta prioridad por contener más de 7 palabras.

Considerando lo anterior el modelo de prioridad basado en diccionario fue capaz de identificar el 70% de mensajes de interés para la institución.

En la siguiente sección se muestran gráficos que facilitan la comprensión de los resultados para un usuario que requiera emplear los modelos aquí descritos.

A continuación se presentan los gráficos utilizados para mostrar a un usuario el resultado del procesamiento de los modelos.

3.6.3 Visualización del Modelo de clasificación de polaridad

Para mostrar a un usuario del modelo las proporciones de polaridad emocional de un conjunto de mensajes se emplean gráficos de pastel. Se consideran los casos de clasificación para 3 clases (positivo, negativo y neutral) y para 2 clases (positivo y negativo).

A partir de la predicción del clasificador, asignada a un arreglo, y usando la función *pyplot.pie* de la librería *mathplotlib*, se generan gráficos de pastel para visualizar la proporción de mensajes clasificados como positivos, negativos o neutrales.

En los Gráficos 2 y 3 se muestran los gráficos de proporciones de polaridad emocional generados por el clasificador entrenado con la concatenación de la BD TASS y la BD Propia, para 2 y 3 clases.

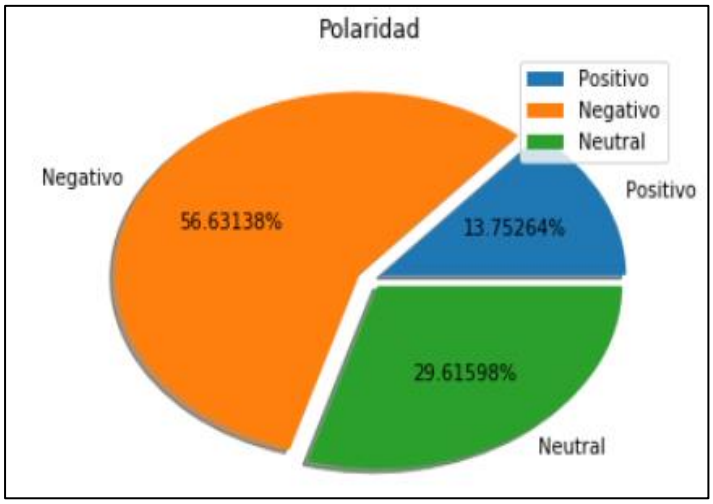


Gráfico 2. Distribución proporcional de la clasificación, 3 clases, BD TASS + BD Propia. Fuente: elaboración propia.

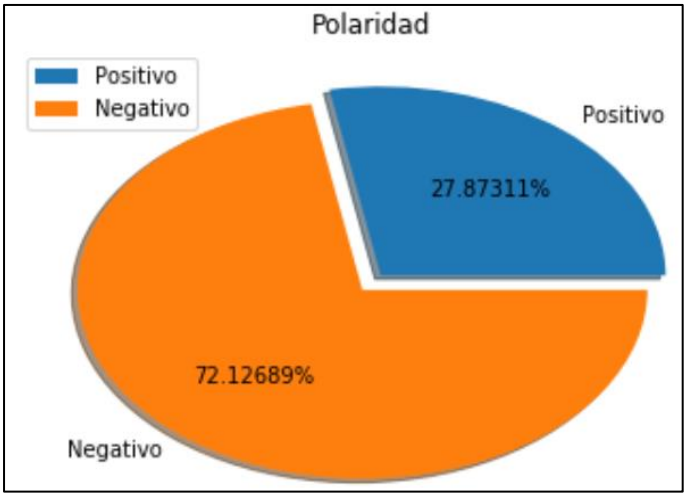


Gráfico 3. Distribución proporcional de la clasificación, 2 clases, BD TASS + BD Propia. Fuente: elaboración propia.

3.6.4 Visualización Modelo de prioridad de mensajes basado en diccionario

Utilizando la función *pyplot* de la librería *mathplotlib,* a partir del arreglo de frecuencias de términos que resulta del modelo, se genera el gráfico de Frecuencia de Términos de Diccionario para los mensajes que conforman el corpus.

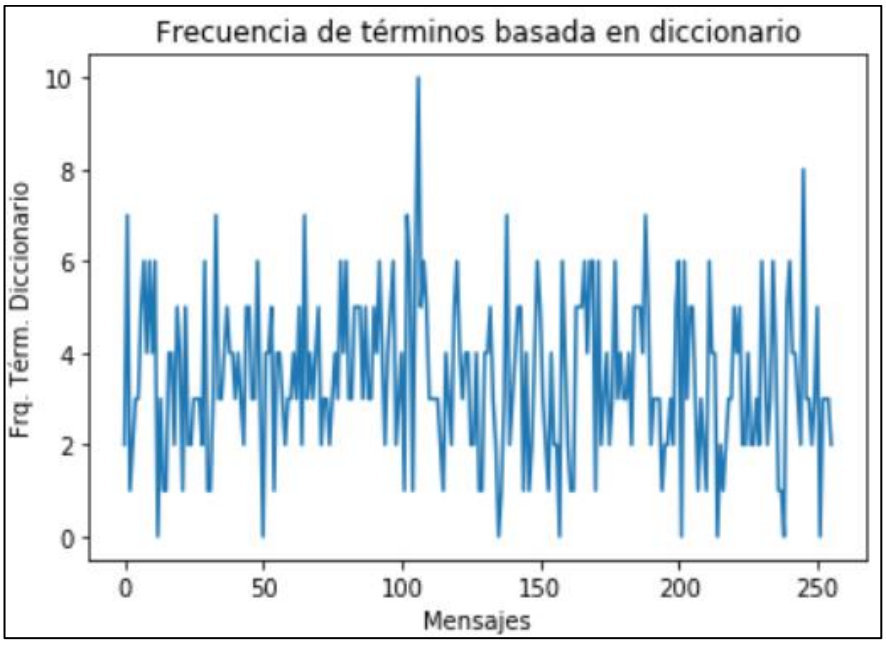


Gráfico 4. Gráfico de Frecuencia de términos de diccionario en Mensajes de BD. Fuente: elaboración propia

El gráfico 4 muestra la frecuencia de términos significativos basados en diccionario para cada uno de los mensajes que integran un conjunto de prueba, particularmente se muestra el gráfico del conjunto de mensajes con los que se generó el diccionario.

En la siguiente sección se describen los resultados de la implementación de los modelos de clasificación de la polaridad emocional y prioridad de mensajes basada en frecuencia de términos de diccionario, en ambos casos empleando ejemplos de mensajes adicionales a los que en esta sección se han presentado.

Capítulo 4.

Resultados

Capítulo 4. Resultados

En este capítulo se presentan los resultados de la implementación de modelos para los dos casos que se han analizado: clasificación de polaridad y prioridad de mensaje basada en frecuencia de diccionario.

Para ambos casos se emplean bases de datos de mensajes que no se utilizaron para el análisis y que fueron brevemente descritas en la sección 3.2.4.

A continuación, en las secciones 4.1 y 4.2 se abordan la descripción de los resultados de implementar cada uno de los dos casos.

4.1 Implementación del modelo de clasificación de polaridad

De acuerdo a los resultados de la evaluación de los clasificadores, presentado en la sección 3.5.1, el algoritmo de clasificación de Máquinas de Vectores de Soporte es capaz de generar predicciones más representativas respecto al resto de los algoritmos, particularmente cuando el arreglo de tokens es vectorizado con TF-IDF. En la misma sección se mostraron los resultados de las métricas de evaluación para distintos casos de clasificación usando la Base de Datos TASS, la BD Propia y una concatenación de ambas bases, sin que ninguno de los clasificadores presente un mejor desempeño respecto a los demás.

Considerando lo antes descrito se realiza la implementación de dos clasificadores de aprendizaje supervisado, con algoritmo de Máquina de Vectores de Soporte, entrenados con una base de datos conjunta integrada a partir de la base TASS y la base de diseño propio, para 3 y 2 clases.

La implementación se realizó con lenguaje Python en la plataforma Jupyter, misma que puede consultarse en los Anexos de este documento, conforme al proceso de clasificación de polaridad que se describe a continuación.

4.1.1 Proceso de clasificación de polaridad

Para realizar la clasificación de la polaridad de un conjunto de mensajes con aprendizaje supervisado se emplean un conjunto de datos de entrenamiento etiquetados y un conjunto de prueba para el cual se desea conocer su polaridad.

La Figura 2 muestra el diagrama de bloques del proceso para clasificar la polaridad de un conjunto de mensajes.

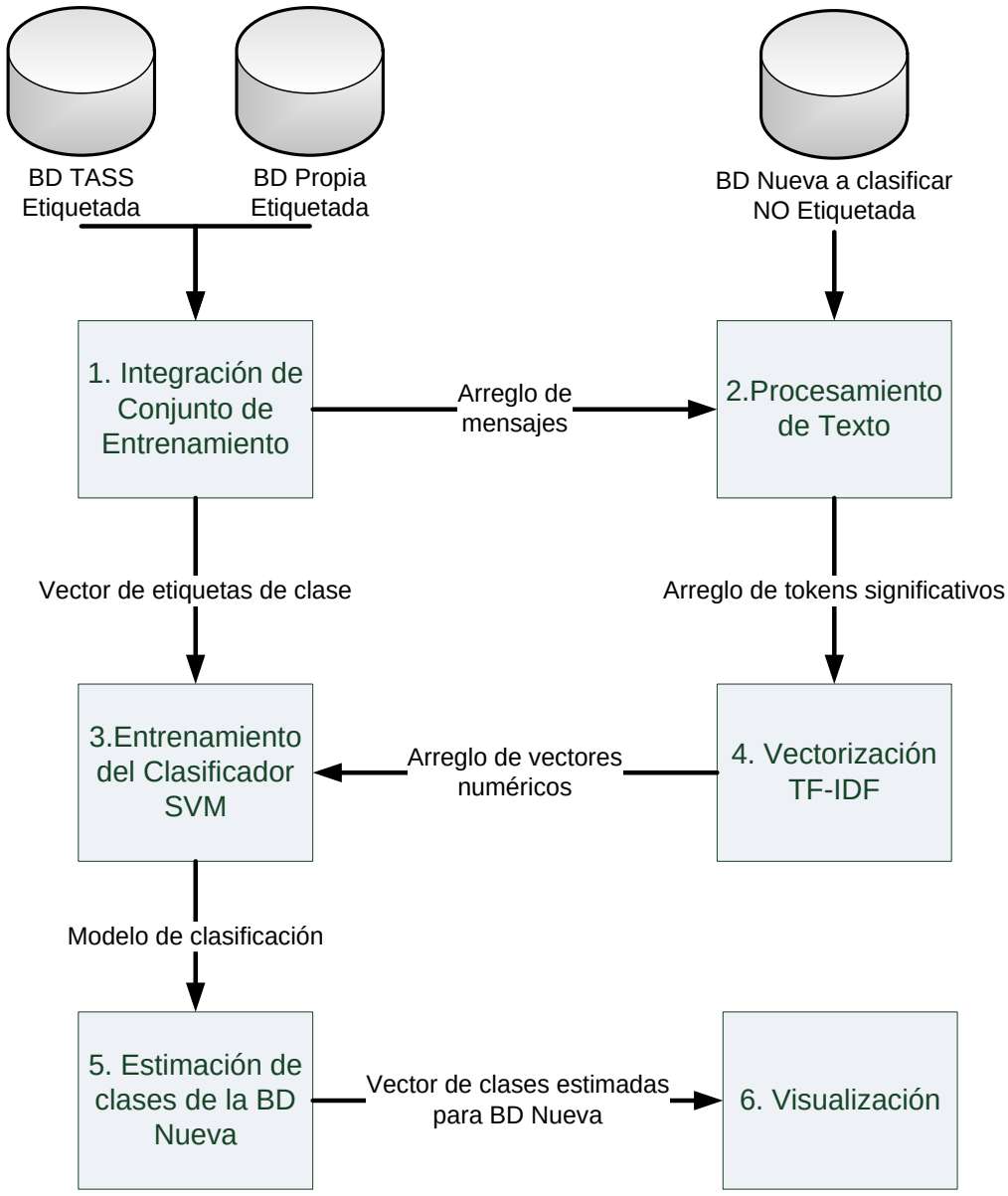


Figura 2. Proceso para clasificación de la polaridad emocional de mensajes. Fuente: elaboración propia.

El proceso de clasificación de polaridad con aprendizaje supervisado consta de 6 pasos que a continuación se describen:

1. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base de datos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).

2. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación y de palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.

3. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en un vector numérico para su procesamiento.

4. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con los mensajes vectorizados y su vector de clases.

5. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica en la predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.

6. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadas con el clasificador.

4.1.2 Resultados de clasificación de polaridad emocional

El proceso de clasificación antes descrito se implementó con Python en Jupyter y se ejecutó para dos bases de datos de mensajes coyunturales que fueron recuperados usando la API de Twitter con los hashtag #PrensaProstituida (14,649 twits) y #CobardeMatoncito (65,406 twits).

Para cada base de datos se realizó clasificación para 2 y 3 clases de polaridad emocional. Las Gráficas 5 y 6, muestran la distribución proporcional de la polaridad emocional para la base de datos de #CobardeMatoncito. El tiempo de procesamiento empleado para clasificar 2 clases fue de 84 minutos y para 3 clases fue de 183 minutos, empleando un computadora de escritorio con procesador Intel Core I7, SO Windows 64, 16 GB-RAM.

La coyuntura #CobardeMatoncito surgió por críticas al Jefe del Estado Mexicano por parte de funcionarios bolivianos a finales de diciembre de 2019, rápidamente el tema se hizo tendencia en México y Latinoamerica, resultó de interés conocer de manera general la carga emocional que se estaba empleando

en los mensajes, misma que de acuerdo a los resultados mostrados en las gráficas es mayoritariamente negativa.

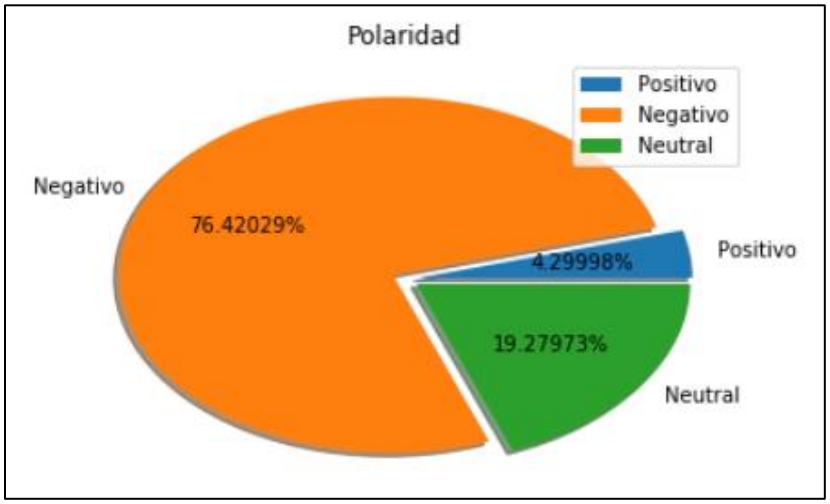


Gráfico 5. Clasificación de la BD #CobardeMantoncito para 3 clases. Procesada en 182 minutos.
Fuente: elaboración propia.

Los Gráficos 7 y 8 muestran la distribución proporcional de la polaridad emocional para la base de datos de #PrensaProstituida. El tiempo de procesamiento empleado para clasificar 2 clases fue de 9 minutos y para 3 clases fue de 22 minutos, empleando un computadora de escritorio con procesador Intel Core I7, SO Windows 64, 16 GB-RAM.

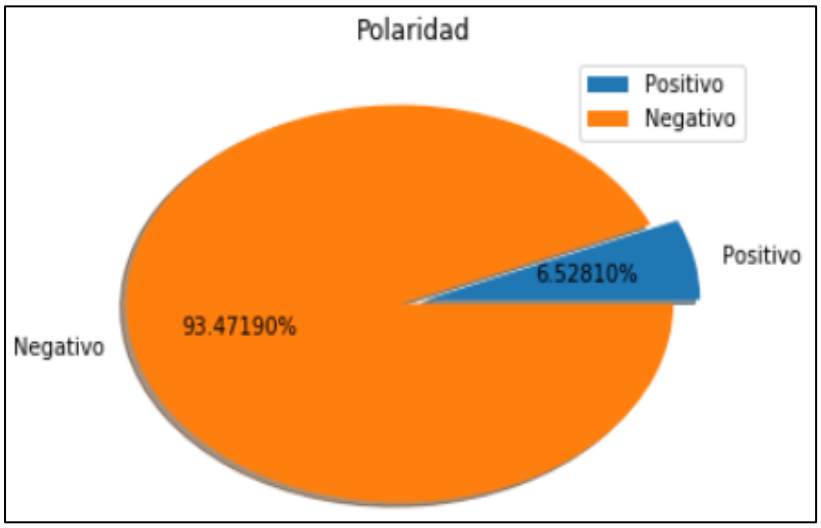


Gráfico 6. Clasificación de la BD #CobardeMantoncito para 2 clases. Procesada en 84 minutos.
Fuente: elaboración propia.

La coyuntura #PrensaProstituida surgió a principios de noviembre de 2019 como resultado de declaraciones poco favorables del Jefe del Estado Mexicano al referirse de esta manera a algunos periodistas, el tópico pronto se hizo tendencia y es de interés conocer de manera general la carga emocional empleada en los mensajes. De acuerdo a los resultados obtenidos, la carga emocional de los mensajes es negativa.

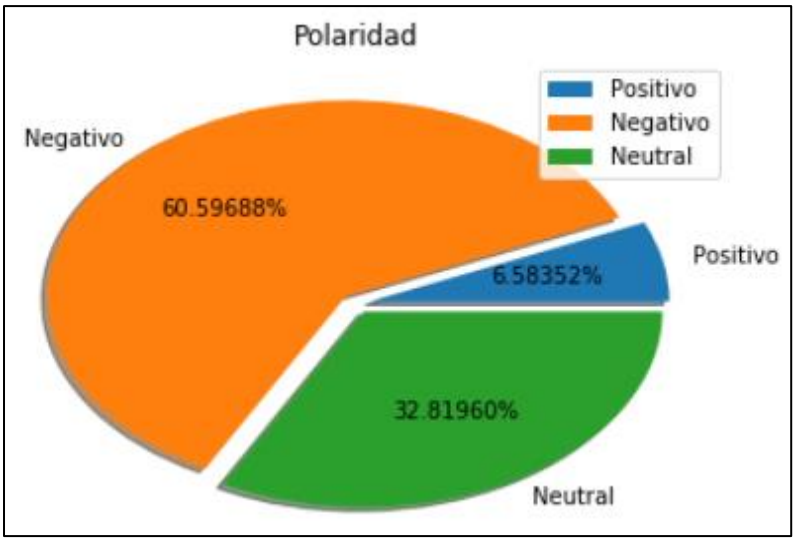


Gráfico 7. Clasificación de la BD #PrensaProstituida para 3 clases. Procesada en 22 minutos.
Fuente: elaboración propia.

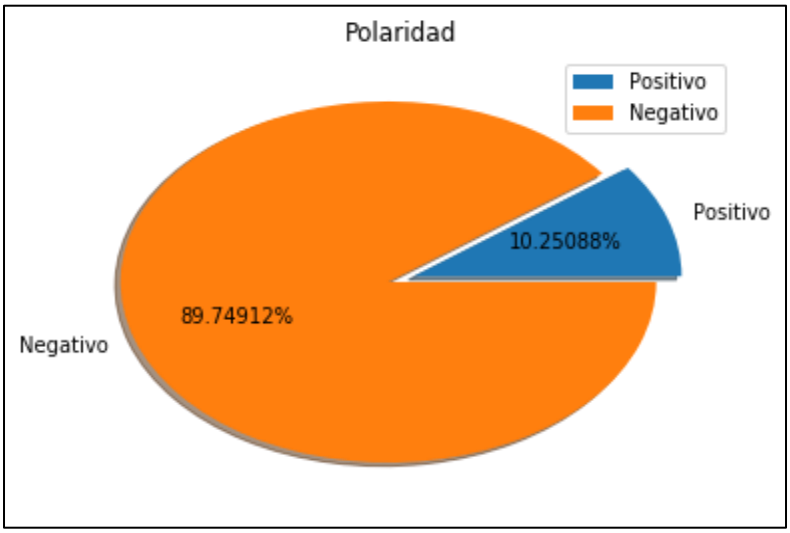


Gráfico 8. Clasificación de la BD #PrensaProstituida para 2 clases. Procesada en 9 minutos.
Fuente: elaboración propia.

Para la clasificación de ambas bases de datos puede observarse que la proporción de twits clasificados como neutrales con el clasificador de 3 clases, son clasificados como Negativos en el clasificador de 2 clases.

4.2 Implementación de Modelo de prioridad de mensajes basado en diccionario

Este modelo de identificación de mensajes prioritarios con base en la frecuencia de términos de un diccionario, se implementó con Python en plataforma Jupyter.

A continuación se describe el proceso para identificar mensajes prioritarios así como los resultados de su implementación en una base de mensajes de prueba.

4.2.1 Proceso para prioridad de mensajes basado en diccionario

En la figura 3 se muestra el diagrama de bloques que describe gráficamente el proceso para identificar mensajes prioritarios, basado en un diccionario de palabras significativas. Los pasos considerados en dicho procesos son los siguientes:

1. Procesamiento de texto, para generar tokens limpios a partir de la base de datos de mensajes significativos.

2. Generación de Diccionario. Utilizando la función vecorizer.vocabulary_ de la librería feature_extraction.text.CountVectorizer de ScyKit-Learn, se obtiene el diccionario de términos contenidos en la base de datos. Posteriormente se realiza una revisión y eliminación de palabras poco significativas para contar con un diccionario final. Para este caso a partir de una base de datos de 255 mensajes, proporcionados por la Institución, se obtuvo un diccionario de 555 palabras significativas.

3. Procesamiento de texto. Para generar tokens limpios de una base de datos de mensajes para la cual se desea identificar los mensajes prioritarios.

4. Modelo de Frecuencia de término. Para cada uno de los vectores de tokens de la base de datos que se desea clasificar, se identifican y cuentan las palabras del diccionario que contenga el mensaje; mientras más palabras significativas contenga un mensaje se le considera más prioritario.

5. Visualización y Consulta. Se muestra gráfico de frecuencia de términos para la base de mensajes. Usando funciones para gestión de arreglos es posible realizar consultas para recuperar los mensajes que contengan más palabras significativas, los cuales son considerados de mayor prioridad.

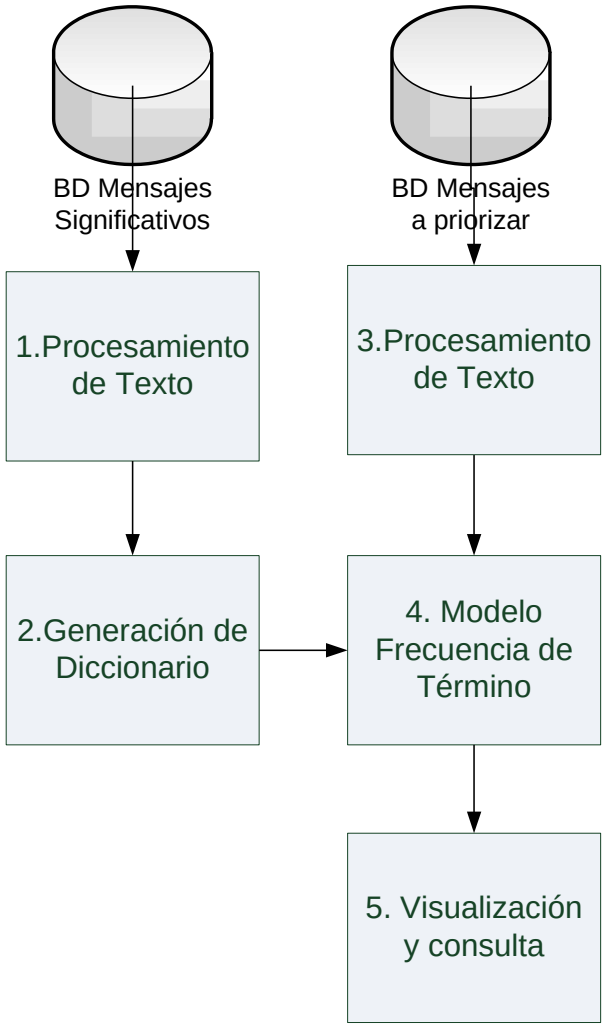


Figura 3. Diagrama del modelo de frecuencia de término basada en diccionario. Fuente: elaboración propia.

4.2.2. Resultados del Modelo de prioridad basado en diccionario

A continuación se muestra el resultado de la implementación del modelo de identificación de mensajes prioritarios basado en diccionario, para tal efecto se usa como ejemplo una base de datos de 3717 mensajes de Twitter, recuperados entre

el 1 y el 7 de enero de 2020 y que contenían o hacían mención del usuario @semar_mx.

El Gráfico 9 muestra la cantidad de mensajes identificados para distintas frecuencias significativas de término.

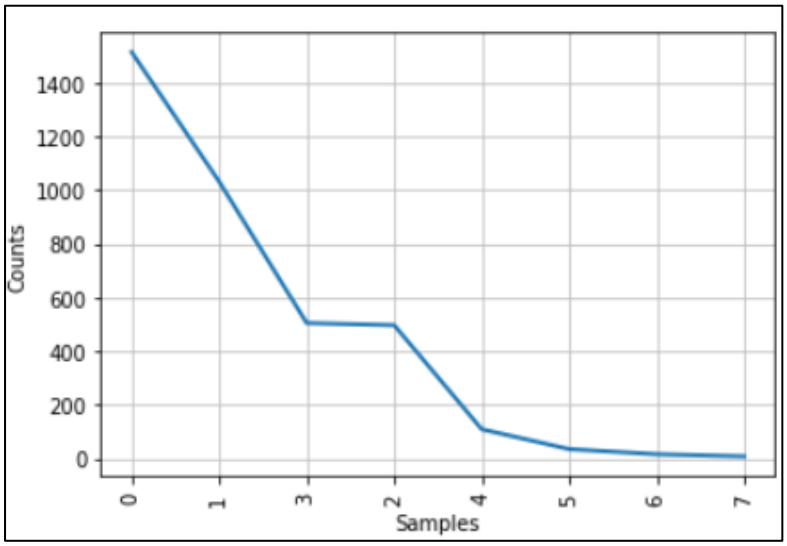


Gráfico 9. Cantidad de mensajes para cada frecuencia de término significativo.
Fuente: elaboración propia.

De las consultas realizadas a la implementación (Ver Anexos) se obtiene el tabulador mostrado en el Cuadro 11, del cual se puede leer que 1515 mensajes no tiene ninguna relevancia pues no contienen ningún término del diccionario, 2143 mensajes contienen entre 1 y 4 términos del diccionario, y 59 mensajes contienen más de 4 términos significativos.

Cantidad de mensajes para cada frecuencia de término significativo
[(0, 1515), (1, 1032), (3, 505), (2, 497), (4, 109), (5, 35), (6, 16), (7, 8)]

Cuadro 11. Cantidad de mensajes para cada frecuencia de término significativo.
Fuente: elaboración propia.

Puede concluirse que de los 3717 mensajes recuperados en los primeros 7 días del año, sólo 59 requieren atención prioritaria debido a que contienen una cantidad significativa de términos del diccionario.

En este capítulo se han presentado los resultados de la implementación en Python de dos modelos para la identificación de mensajes prioritarios desde

dos enfoques: clasificación de la polaridad usando aprendizaje supervisado y prioridad de mensaje a partir de frecuencia de palabras significativas en un diccionario.

4.3 Implementación de Modelo híbrido de prioridad de mensajes

El modelo híbrido de identificación de mensajes prioritarios se basa en la coordinación secuencial del enfoque de análisis de polaridad de sentimientos comentado en 4.1 y el enfoque de frecuencia de palabras significativas de un diccionario descrito en la sección 4.2. Este enfoque se plantea debido a que la polaridad de sentimientos puede reflejar la intención de uno o más usuarios de ejercer alguna acción que puede afectar no sólo la opinión del la institución a la que pertenece el DAI, sino también a su desempeño, en función del interés del DAI cualquiera de los conjuntos de mensajes clasificados como positivo o negativo podría analizarse para identificar mensajes prioritarios a partir de la frecuencia de palabras significativas del diccionario. A continuación se describe el proceso para identificar mensajes prioritarios así como los resultados de su implementación en una base de mensajes de prueba.

4.3.1. Proceso para prioridad de mensajes basado en modelo híbrido

En la figura 4 se muestra el diagrama de bloques que describe gráficamente el proceso para identificar mensajes prioritarios, basado en el modelo híbrido. Los pasos considerados en dicho proceso son los siguientes:

1. Clasificación de polaridad. Se realiza la clasificación de polaridad de una base de datos nueva según lo indicado en la sección 4.1.1.

2. Extracción de mensajes de la clase de interés. En función del criterio del analista, determina la clase de interés (positiva o negativa) y se realiza la extracción mensajes clasificados, asignándolos a un arreglo para su análisis.

3. Modelo de frecuencia de término basada en Diccionario. Empleando el modelo descrito en 4.2, se realiza la identificación de palabras significativas y el ordenamiento de mensajes de la clase seleccionada (positiva o negativa).

4. Visualización y consulta. Al igual que en los modelos descritos en 4.1 y 4.2, se muestra el gráfico de proporción de polaridad y el gráfico de frecuencia de términos para la base de mensajes. Usando funciones para gestión de arreglos es

posible realizar consultas para recuperar los mensajes que contengan más palabras significativas, los cuales son considerados de mayor prioridad.

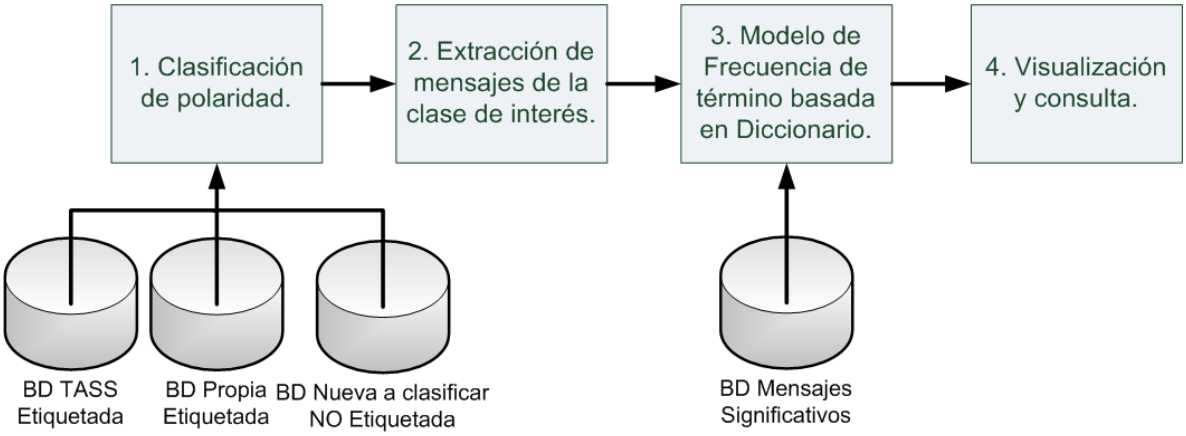


Figura 4. Diagrama del modelo híbrido para identificación de mensajes prioritarios. Fuente: Elaboración propia.

4.3.2. Resultados del Modelo híbrido de prioridad basado de mensajes

A continuación se muestra el resultado de la implementación del modelo híbrido de identificación de mensajes prioritarios, para tal efecto se usa como ejemplo la misma base de datos empleada en la sección 4.2, la cual consta de 3717 mensajes de Twitter, recuperados entre el 1 y el 7 de enero de 2020 y que hace mención del usuario @SEMAR_MX.

Primero se realiza el análisis de polaridad con el modelo de clasificación supervisada empleando aprendizaje computacional descrito en la sección 4.1; el clasificador generó los conjuntos de mensajes positivos y negativos en 2 minutos; el Gráfico 10 muestra las proporciones para cada clase.

Suponiendo con un analista decide que la clase de interés es la negativa, se extrae el conjunto de mensajes clasificados con dicha polaridad. Una vez extraídos se emplea el modelo de prioridad basado en frecuencia de diccionario descrito en la sección 4.2.

El Gráfico 11 muestra el gráfico de número de palabras significativas respectos a cantidad de mensajes que las contienen, el detalle de las cantidades puede revisarse en el Cuadro 12, en el cual se muestran pares ordenados, siendo

el número de palabras significativas para las abscisas y el número de mensajes para las ordenadas.

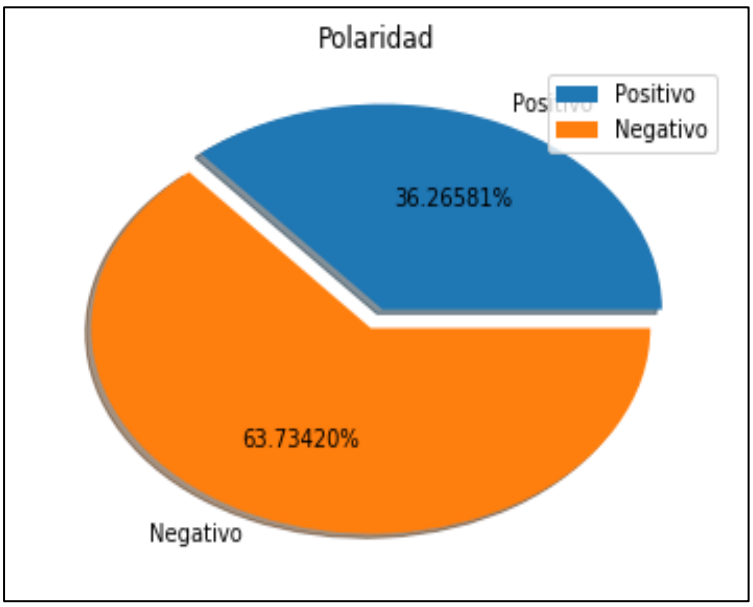


Gráfico 10. Distribución porcentual de la polaridad en mensajes que mencionan al usuario @semar_mx.
Fuente: Elaboración propia.

La clase negativa que se analiza está compuesta de 2769 mensajes que corresponden al 63.7% del total de mensajes, de éstos 41 mensajes contienen más de 5 palabras significativas que, respecto al análisis de la base de mensajes sin clasificación de polaridad presentado en la sección 4.2, representan 18 mensajes menos con la misma cantidad de palabras significativas (Ver Cuadro 11 y comparar con Cuadro 12), esta diferencia reduce la cantidad de mensajes a revisar por el analista, priorizando aquellos con más palabras significativas y orientación negativa hacia el objeto de estudio.

Cantidad de mensajes para cada frecuencia de término significativo
[(0, 1186), (1, 591), (2, 324), (3, 152), (4, 75), (5, 25), (6, 11), (7, 5)]

Cuadro 12. Cantidad de mensajes para cada frecuencia de término significativo de la clase negativa.
Fuente: elaboración propia.

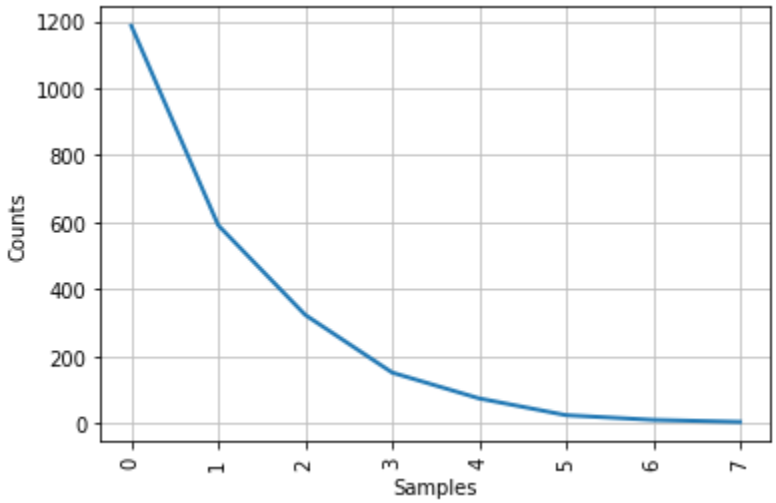


Gráfico 11. Cantidad de mensajes para cada frecuencia de término significativo de la clase negativa.
Fuente: elaboración propia.

Con el objetivo de comparar los resultados a nivel de contenido detallado de los mensajes, en el Cuadro 13 se muestran los 3 primeros mensajes priorizados con enfoque de frecuencia de diccionario y con enfoque híbrido para la clase negativa.

Los 3 primeros mensajes priorizados con el modelo basado en diccionario presentan una redacción positiva o informativa mientras que los mensajes priorizados con el modelo híbrido describen eventos o actos, con uso de palabras significativas y connotación negativa (ataque, dispararon, golpeado, morir), que podrían brindar información sobre eventos de interés al desempeño del DAI.

Se considera que la implementación del modelo híbrido mejora el proceso que actualmente realiza al DAI, pues en función del contexto que se analice, la polaridad de un mensaje combinado con palabras significativas podría tener mayor relevancia para identificar eventos que afecten no sólo la imagen institucional sino su desempeño.

Para la base de datos analizada, de acuerdo al Cuadro 13, el modelo híbrido prioriza mensajes de eventos delictivos que aportan información que podría poner en riesgo la eficacia de las funciones de la Institución, mientras que el modelo basado en diccionario prioriza mensajes informativos o positivos que podrían no ser un riesgo a las funciones.

En los Anexos puede revisarse la implementación del modelo de clasificación de polaridad para los conjuntos de mensajes #CobardeMatoncito y #PrensaProstituida, así como la implementación del modelo de prioridad basada en diccionario así como el Modelo híbrido para el conjunto de mensajes relativos al usuario @SEMAR_mx.

En el siguiente capítulo se abordan las conclusiones finales de este trabajo así como la reflexión respecto al alcance y las limitaciones identificadas en el curso del desarrollo de la solución estratégica descrita en este documento

Primeros 3 Mensajes priorizados Modelo basado en diccionario Conjunto completo	Primeros 3 Mensajes priorizado Modelo híbrido usando Clase Negativa
@SEMAR_mx #FuerzaMexico Organización Nacional de Civiles y Militares A.C. Felicita efusivamente a los miembros de la *Marina Armada de México con motivo del #AnoNuevo2020 Confiando con profunda convicción en su patriotismo y espíritu de servicio. !La Patria es Primero! #SoyNaval https://t.co/kW8BBdfIY7 7	Detienen elementos de la @SEMAR_mx a presunto jefe del CJNG, ligado al ataque al bar Caballo Blanco ubicado en Coatzacoalcos. Junto con él fue detenida otra persona, ambos portaban bolsas con dosis de droga #Veracruz #verfollow https://t.co/cRXBAWFlDs 7
@lopezobrador_ @Pemex @SEDENAmx @SEMAR_mx @GN_MEXICO_ @STPS_mx #ConferenciaAMLO El presidente @lopezobrador_ compara el juicio de Genaro García Luna con #Odebrecht y señala que es más importante porque en el caso del exsecretario de Seguridad Pública hay muchas vidas perdidas y desapariciones involucradas https://t.co/iEdMsXLiul https://t.co/RjEYn8suZH 7	#GUAYMAS Hombres armados dispararon esta noche contra un elemento de la policí-a municipal. El agente recibió al llegar a su casa 8 impactos de bala. Se trata de Eduardo Manjarrez, quien hace un mes fue golpeado por elementos de @SEMAR_mx https://t.co/snvpam8AeC 7
@lopezobrador_ @Pemex @SEDENAmx @SEMAR_mx @GN_MEXICO_ #ConferenciaAMLO @lopezobrador_ rechaza que Romero Deschamps siga controlando el Sindicato Petrolero. Luisa Marí-a Alcalde, secretaria de la @STPS_mx asegura que "El próximo secretario general del #STPRM, será a través del voto libre, directo y secreto" https://t.co/iEdMsXLiul https://t.co/OTRL7YK7PY 7	@el_pais Aquí el jefe de Genaro Garcia Luna el NARCO A CARGO DE LA SEGURIDAD DE MEXICO POR 6 AÑOS Su jefe el Narco FELIPE CALDERON COBRO MILLONES DE DOLARES dando protección y atacando a los enemigos del Cartel de Sinaloa MANDO A MORIR CIVILES SOLDADOS Y POLICÍAS @SEDENAmx @SEMAR_mx https://t.co/vCkff3PwRr 7

Cuadro 13. Mensajes priorizados con el modelo basado en diccionario (Izq.) y el modelo híbrido (Der.).
Fuente: Elaboración propia.

Conclusiones

Conclusiones

Las redes sociales se han convertido en fuentes de información colectiva, los usuarios de éstas exponen sus ideas y opiniones respecto a un objeto, persona, u organización, también describen lo que está ocurriendo en su entorno e interactúan con los comentarios de otros usuarios.

La red social de mensajería Twitter es un microblog muy atractivo no sólo para los usuarios orgánicos que publican o consultan opiniones y eventos, lo es también para las instituciones de gobierno que pueden emplearlo como una plataforma de propaganda política, como un medio para conocer la opinión de la población y para identificar eventos y protagonistas de la sociedad que pueden afectar el contexto histórico; la oportuna identificación y análisis de las opiniones y eventos que mencionan los usuarios de Twitter puede ayudar a las Instituciones de Gobierno a favorecer el diálogo con la población, a mejorar su desempeño y a implementar estrategias para mantener la confianza con los ciudadanos.

El trabajo presentado en esta tesis surge de la necesidad del Departamento de Análisis de Información de una Institución de Gobierno para identificar eficazmente mensajes en Twitter que son prioritarios a su propósito, esta actividad se realiza cotidianamente de forma manual desde dos enfoques: el análisis de la polaridad de un mensaje o conjunto de mensajes para evitar o contener afectaciones mediáticas y la identificación de mensajes que por su contenido informan sobre situaciones que podrían afectar el desempeño de sus funciones.

Para realizar su labor el personal del DAI cuenta con la capacidad de recopilar y analizar hasta 1000 mensajes por jornada de trabajo, los cuales son leídos y clasificados manualmente utilizando como criterio de decisión el discernimiento del analista a cargo. La demanda de analizar muchos más mensajes cada día ha planteado la necesidad de mejorar la eficacia de su proceso empleando técnicas de análisis automático.

Con el desarrollo de esta tesis, al implementar métodos automáticos para la clasificación de la polaridad, se demuestra que la eficacia en el proceso de identificación de mensajes prioritarios se supera en gran medida, pues la capacidad de análisis manual del DAI es de 1000 mensajes por jornada de trabajo

(8 horas) y de acuerdo a los resultados presentados en la sección 4.1, es posible clasificar más de 65 mil mensajes en 183 minutos (3 horas), permitiendo al personal del DAI aumentar su capacidad de análisis y emplear su tiempo en labores más estratégicas que operativas. De manera similar, el tiempo de análisis para clasificar más de 3000 mensajes por su contenido de palabras significativas, conforme al modelo de frecuencia de diccionario presentado en la sección 4.2, es menor a 10 minutos. Ambos casos mejoran la eficacia del proceso de identificación de mensajes prioritarios del DAI.

En relación al modelo de clasificación de la polaridad, se implementaron en Python-Jupyter los algoritmos de clasificación supervisada de la librería SciKit-Learn: Máquinas de Vectores de Soporte, Naive Bayes, Centroide más cercano y Árboles de decisión. Para los conjuntos de entrenamiento y prueba se emplearon la base de datos InterTASS de la Sociedad Española para el Procesamiento de Lenguaje Natural, una base de datos de diseño propio, etiquetada por 5 voluntarios, y una concatenación de ambas bases para conformar un tercer conjunto de mensajes más grande. Las tres bases de datos se acondicionaron para 3 clases (Positivo, Negativo y Neutral) y para 2 clases (Positivo y Negativo), de esta manera se contó con 6 conjuntos para entrenamiento y prueba de los clasificadores.

Los mensajes de las bases de datos fueron vectorizados para su procesamiento usando TF (Frecuencia de Término) para evaluar un primer caso y con TF-IDF (Frecuencia Inversa de Documento) para evaluar un segundo caso.

Los resultados de los experimentos de clasificación, mismos que se pueden consultar en la sección 3.4.1, muestran que los clasificadores presentan mejor desempeño con el algoritmo de Máquinas de Vectores de Soporte con entrada de arreglo de mensajes vectorizados con TF-IDF, particularmente la clasificación con las clases Positiva y Negativa es más precisa que la clasificación para tres clases (positivo, negativo y neutral). Las bases de datos InterTASS y la base de datos propia fueron evaluadas de forma cruzada para entrenamiento y prueba sin que alguna de las dos presentara un mejor desempeño lo cual indica que el procedimiento empleado para el etiquetado basado en discernimiento humano es

comparable con la base de mensajes InterTASS elaborada por la Sociedad Española de Lenguaje Natural; posteriormente ambas bases se concatenaron y su desempeño como conjunto de entrenamiento no presentó alguna ventaja en las medidas de desempeño al compararla con los resultados de entrenamiento de cada base de forma individual. Aunque la concatenación de ambas bases no mejora la precisión del clasificador, se considera que tener un conjunto de entrenamiento mayor puede generar un clasificador entrenado para más casos y ser más representativo.

Los experimentos realizados pueden consultarse en los Anexos de este documento.

El modelo de clasificación con algoritmo de Máquinas de Vectores de Soporte y vectorización de mensajes con TF-IDF fue implementado para su uso como aplicación en un NoteBook de Jupyter, en este trabajo fueron presentados en la sección 4.1 dos casos coyunturales para clasificación de polaridad emocional: un conjunto de mensajes con el hashtag #CobardeMatoncito y otro con #PrensaProstituida. Ambos casos presentaron significativa carga negativa en la clasificación para 3 y 2 clases y pudo observarse, sin comprobarlo, que los mensajes clasificados como neutros en el clasificador de 3 clases, fueron propensos a clasificarse como negativos en el clasificador de 2 clases. La implementación y resultados de los modelos para estos casos pueden consultarse en los Anexos. La eficacia en términos de cantidad de mensajes analizados respecto al tiempo de procesamiento es la siguiente para cada uno de las pruebas: PrensaProstituida, 3 clases, 14649 mensajes/22 minutos; 2 clases, 14649 mensajes/9 minutos; CobardeMatoncito, 3 clases, 65406 mensajes/183 minutos; 2 clases, 65406 mensajes/84 minutos; esta eficacia evidentemente supera por mucho la actual en el DAI de 1000 mensajes/480 minutos.

Respecto a la identificación de mensajes prioritarios, se propuso e implementó un modelo de identificación de mensajes basada en diccionario de términos; para tal efecto se contó con una pequeña base de datos de mensajes que previamente fueron identificados como casos de interés por parte de DAI, a partir de ésta se construyó un diccionario de 555 palabras significativos y se

implementó un modelo para recuperar y contar la frecuencia de términos del diccionario contenidos en los mensajes de la base de datos, tal como se explica en la sección 3.5.2.

El modelo de identificación de mensajes prioritarios se basa en la frecuencia de términos significativos del diccionario, es decir se considera que un mensaje es de mayor prioridad respecto a otro si contiene más palabras significativas del diccionario. El modelo fue probado inicialmente con la base de datos que dio origen al diccionario, posteriormente se implementó en software, usando NoteBook de Jupyter, para su uso como prototipo experimental en la identificación de mensajes. En la sección 4.2 se describen los resultados de su aplicación a una base de datos de 3717 mensajes de Twitter que mencionaron a la cuenta @semar_mx entre el 1o y el 7 de enero de 2020, los resultados indican que más del 50% de los mensajes pueden descartarse debido a la ausencia de términos significativos en su contenido, y tan solo 59 mensajes (1.6% del total) se consideran de alta prioridad debido a que presentan mayor frecuencia de términos del diccionario. La implementación y los resultados del procesamiento de este caso puede consultarse en el Anexos.

Como se comentó, el DAI emplea dos enfoques independientes para la identificación de mensajes prioritarios: el análisis de polaridad para identificar amenazas mediáticas y el contenido de palabras que describen eventos que podrían afectar el desempeño institucional. Sin embargo no evalúa la presencia de palabras significativas en los conjuntos de mensajes analizados en polaridad y que podrían contener evidencia de la intención de algún usuario o grupos de usuarios de realizar algún acto, o ser testigos de alguno y comentarlo, tal que pudiera afectar el desempeño institucional. Esto motivó la coordinación, en un modelo híbrido, de los enfoques de análisis de polaridad y de frecuencia basado en diccionario para identificar mensajes prioritarios, el cual fue implementado en un Notebook de Jupyter y se probó empleando la base de datos de 3717 mensajes relativos al usuario @semar_mx y se comparó con los resultados del modelo basado en diccionario.

De acuerdo a lo reportado en la sección 4.3, luego de clasificar la polaridad de los mensajes y evaluar la clase negativa con el modelo de frecuencia basada en diccionario de palabras significativas se observa que los primeros mensajes priorizados están relacionados a eventos delictivos que podrían afectar el desempeño, en este caso del usuario @semar_mx, mientras que los primeros mensajes priorizados con todos los mensajes son relativos a información de otras instituciones, lo anterior aporta indicios respecto a que la clase negativa analizada con el modelo de prioridad basada en diccionario permite identificar con mayor eficacia mensajes prioritarios que pueden afectar el desempeño de la institución a la que pertenece el DAI, esto al discriminar con dos métodos los mensajes con menor contenido de interés institucional.

En el ejemplo analizado para la base de datos de 3717 mensajes, usando sólo el modelo basado en diccionario se identificaron 59 mensajes prioritarios y usando el modelo híbrido se identificaron 41, es decir se descartaron 18 mensajes que en bases de datos mayore podría representar una cifra mucho mayor.

Considerando lo antes descrito, en el presente trabajo se presentó un primer alcance tecnológico que contribuye a mejorar el monitoreo de redes sociales con fines de seguridad e integridad institucional para el DAI, lo anterior mediante la identificación de mensajes prioritarios de Twitter usando análisis de polaridad de temas de interés en el contexto institucional así como identificando palabras significativas de un diccionario creado para ese fin. El alcance tecnológico está limitado a tres implementaciones en NoteBook de Jupyter que pueden ser operadas fácilmente por un analista de redes sociales, la primera corresponde a la clasificación automática de la polaridad empleando aprendizaje supervisado, la segunda a la identificación de mensajes prioritarios a partir de la frecuencia de palabras de un diccionario y la tercera una combinación de ambas.

Los modelos analizados e implementados en NoteBook de Jupyter contribuyen a mejorar la eficacia del proceso de identificación de mensajes prioritarios del DAI, la cantidad de mensajes que pueden ser analizados en pocos minutos es mucho mayor que la capacidad actual limitada por el discernimiento humano, el cual no es sustituido con este trabajo, sino que puede concentrarse en

un análisis más estratégico enfocado en los resultados que arrojan los modelos propuestos. Adicionalmente, la coordinación de los modelos de clasificación de polaridad y prioridad basada en diccionario es un enfoque nuevo para el DAI y que muestra indicios de que mejora la eficacia en la identificación de mensajes prioritarios que podrían tener potencial de afectar su desempeño.

A partir del desarrollo de este trabajo se identificaron áreas de oportunidad que pueden ser abordadas al corto plazo y mediano plazo.

En relación al método de clasificación de polaridad se identifican las siguientes oportunidades:

- El etiquetado inicial de la polaridad del conjunto de entrenamiento puede mejorarse al incluir personal especializado en los temas de interés del DAI.
- El etiquetado de la polaridad del conjunto inicial puede refinarse mediante técnicas de aprendizaje semi supervisado con la participación de especialistas humanos.
- El método de clasificación de la polaridad podría mejorar su desempeño si se incluyen técnicas de aprendizaje no supervisado basadas en un diccionario de términos ponderados en idioma español.

Respecto al método de identificación de mensajes prioritarios basado en diccionario se identifican las siguientes oportunidades:

- El diccionario puede mejorarse al asignar pesos, ponderaciones o valores específicos a cada uno de sus términos, esto permitirá desarrollar un modelo de prioridad basado en ponderación y no sólo en frecuencia de término.
- El análisis para generar el diccionario debe realizarse periódicamente, pues las palabras significativas podrían resultar obsoletas en el futuro debido a cambios en el contexto histórico nacional.

Respecto al proceso de monitoreo de redes sociales que realiza la institución, se identifican las siguientes oportunidades de mejora:

- Facilitar mecanismos informáticos para que los analistas de seguridad en redes sociales etiqueten los mensajes con diferentes niveles de prioridad,

tal que sea posible desarrollar nuevo modelos basados en aprendizaje supervisado.

- Desarrollar e implementar un modelo de estimación de propagación de audiencia de un mensaje, tal que permite identificar potencial de tendencias coyunturales antes de que estas ocurran.

Finalmente, el modelo híbrido basado en una coordinación secuencial de análisis de polaridad y frecuencia de términos significativos debe probarse en nuevos escenarios para evaluar su eficacia respecto al proceso que maneja actualmente el DAI.

Bibliografía

(1) Octavio Islas. Estadísticas de Twitter en México. Revista Digital Razón y Palabra. Enero 2020. Consultado en línea en enero de 2020. http://www.razonypalabra.org.mx/espejo/ESPEJO_2011/twitter.html

(2) Cheong M., Lee V. Integrating Web-based Intelligence Retrieval and Decision-making from the Twitter Trends Knowledge Base. Proceedings of the 18th ACM Conference on Information and Knowledge Management: CIKM 2009 Co-Located Workshops, NY, USA, 2009.

(3) Liu B., & Zhang L. A survey of opinion mining and sentiment analysis. In Mining Text Data, pp. 415-463, Springer US. 2012.

(4) Dhiraj Murthy. Twitter: Social Communication in the Twitter Age. Cambridge, UK: Polity Press, 2013.

(5) Campos-Domínguez Eva. Twitter y la comunicación política. El profesional de la información. V. 26, n. 5, pp. 785-793, 2017

(6) A. Tumasjan, T. Sprenger, P. Sandner, I. Welpe. Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. International AAAI Conference on Weblogs and Social Media. 2010.

(7) N. A. Diakopoulos, David A. Shamma. Characterizing debate performance via aggregated Twitter sentiment. Conference on Human Factors in Computing Systems (CHI 2010).

(8) Adam Bermingham, Alan F. Smeaton. On Using Twitter to Monitor Political Sentiment and Predict Election Results. Proceedings of the Workshop on Sentiment Analysis where AI meets Psychology (SAAIP 2011), Chiang Mai, Thailand. Nov. 2011.

(9) Bakliwal A., Foster J., et al. Sentiment Analysis of Political Tweets: Towards an Accurate Classifier. Proceedings of the Workshop on Language in Social Media (LASM 2013). Pages 49–58. Atlanta, Georgia. June 13 2013.

(10) Oliveira, D. J. S., Bermejo, P. H. S., Santos, P. A. Can social media reveal the preferences of voters? A comparison between sentiment analysis and traditional opinion polls. Journal of Information Technology & Politics. 14(1), 34-45. 2017.

(11) P. Burnap, R. Gibson, L. Sloan, R. Southern, M. Williams. 140 Characters to Victory?: Using Twitter to Predict the UK 2015 General Election. May 6, 2015, Available at SSRN: https://ssrn.com/abstract=2603433 or http://dx.doi.org/10.2139/ssrn.2603433

(12) E. Kušen, M. Strembeck. Politics, sentiments, and misinformation: An analysis of the Twitter discussion on the 2016 Austrian Presidential Elections. Online Social Networks and Media 5, pp 37–50, 2018.

(13) Simon Kept. Global Digital Report 2019. We Are Social Ltd. 2019. Consultado en línea en enero de 2020. https://wearesocial.com/global-digital-report-2019

(14) R.S. Goud, P.V. Tikone, S.S. Suryawanshi, D. Nagpal. Twitter Data Sentiment Analysis and Visualization. International Journal of Computer Applications (0975 – 8887). Volume 180 – No. 20. February 2018.

(15) Wilson T., Wiebe J., Hoffmann P. Recognizing contextual polarity in phrase-level sentiment analysis. Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. 2005.

(16) Esuli, A., Sebastiani, F. SENTIWORDNET: A publicly available lexical resource for opinion mining. Proceedings of Language Resources and Evaluation (LREC) 2006. Retrieved January 25, 2006.

(17) Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., Kappas. A Sentiment strength detection in short informal text. Journal of the American Society for Information Science and Technology, 61 (12) 2544-255. 2010.

(18) H.M. Chen, P. C. Franks, L. Evans. Exploring Government Uses of Social Media through Twitter Sentiment Analysis. Journal of Digital Information Management", Vol. 14. Number 5. October 2016.

(19) Y. Watequlis S., Y. Puspitasari. TWITTER DATA MINING FOR SENTIMENT ANALYSIS ON PEOPLES FEEDBACK AGAINST GOVERNMENT PUBLIC POLICY. International Journal of Science and Technology, ISSN 2454-5880, 2017.

(20) Pradany, L. N., Fatichah, C. Analisa Sentimen Kebijakan Pemerintah Pada Konten Twitter Berbahasa Indonesia Menggunakan SVM dan K-Medoid Clustering. SCAN Vol. XI (5966), No.1 ISSN: 1978-0087. Februari 2016.

(21) Silva O.D.J., de Souza B.P.H., Pereira J.R., Aparecida B.D. The application of the sentiment analysis technique in social media as a tool for social management practices at thegovernmental level. Brazilian Journal of Public Administration, Rio de Janeiro 53(1):235-251. Jan.-Feb 2019.

(22) Ceron A., Negri F. Public policies go social: using sentiment analysis to support the action of policy-makers across the policy cycle. Dep. of Social and Political Studies, Università di Milano. 2016.

(23) Hopkins D.J., King G. A method of automated nonparametric content analysis for social science. American Journal of Political Science, n. 54(1), pp. 229–247, 2010.

(24) Villena-Román, J., Lana-Serrano, S., Martínez-Cámara, E., González Cristobal, J.C. TASS - Workshop on Sentiment Analysis at SEPLN. Procesamiento del Lenguaje Natural, 50, 2013. Consultado en línea en agosto de 2020. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4657

(25) R.S. Goud, P.V. Tikone, S.S. Suryawanshi, D. Nagpal. Twitter Data Sentiment Analysis and Visualization. International Journal of Computer Applications (0975 – 8887), Volume 180 – No. 20. February 2018.

(26) Vinodhini G., & Chandrasekaran R. M. Sentiment analysis and opinion mining: a survey. International Journal of Advanced Research in Computer Science and Software Engineering, pp. 282-292, 2012.

(27) Liu B. Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies, Vol. 5, Pages 1-167. 2012.

(28) Sebastiani F. Machine learning in automated text categorization. ACM computing surveys (CSUR), 2002.

(29) Yoon, S., Elhadad, N., & Bakken, S. A practical approach for content mining of tweets. American Journal of Preventive Medicine, 45(1), 122-129, 2013.

(30) Martínez-Cámara, E., García-Cumbreras, M.A., Villena-Román, J., García-Morera, J. TASS 2015 - The Evolution of the Spanish Opinion Mining Systems. Procesamiento del Lenguaje Natural, 56, 2016. Consultado en línea en agosto de 2020. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/5284.

(31) Bird, Steven, Edward Loper and Ewan Klein. Natural Language Processing with Python. O'Reilly Media Inc. Consultado en línea en enero 2020. https://www.nltk.org/

(32) D. Cournapeau, M. Brucher, et. al. Scikit-learn Machine Learning in Python. Consultado en línea en enero 2020. https://scikit-learn.org/

(33) T. Augspurgers et. al. Python Data Analysis Library. Consultado en línea en enero 2020. https://pandas.pydata.org/

(34) T.M. Mitchel. Machine Learning. McGraw-Hill Science/Engineering/Math. March, 1997. ISBN: 0070428077.

(35) Villena-Román, J., García-Morera, J., Lana-Serrano, S., González-Cristóbal, J.C. TASS 2013 - A Second Step in Reputation Analysis in Spanish. Procesamiento del Lenguaje Natural, 52, 2014. Consultado en línea en agosto de 2020. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4901

(36) Villena-Román, J., Martínez-Cámara, E., García-Morera, J. Jiménez-Zafra, S. TASS 2014 - The Challenge of Aspect-based Sentiment Analysis. Procesamiento del Lenguaje Natural, 54, 2015. Consultado en línea en agosto de 2020. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/5095

(37) Graff Guerrero M. Notas de la materia de Aprendizaje Computacional. Maestría en Ciencia de Datos e Información. Centro de Investigación e Innovación en TIC (INFOTEC),.Febrero-Junio 2018.

(38) Graff Guerrero M. Notas de la materia de Minería de Texto. Maestría en Ciencia de Datos e Información. Centro de Investigación e Innovación en TIC (INFOTEC). Agosto - Diciembre 2018.

(39) Téllez Ávila E.S. Notas de la materia de Recuperación de la Información. Maestría en Ciencia de Datos e Información, Centro de Investigación e Innovación en TIC (INFOTEC), Febrero - Julio 2019.

(40) Villalba Osornio S. G. Método probabilista para clasificación de polaridad: Negación e Intensificación en análisis de sentimientos. Tesis de Maestría. INAOE. 2016.

Anexos

Anexos

En las siguientes páginas se muestra el código implementado para realizar el análisis de clasificadores para 3 clases y 2 clases (Anexo I y Anexo II). También se incluye la implementación de los modelos de clasificación de polaridad de las base de datos #CobardeMatoncito para 3 y 2 clases (Anexo III y Anexo IV). Asimismo los resultados de la clasificación de la base de datos #PrensaProstituida para 3 y 2 clases (Anexo V y Anexo VI). Finalmente se presenta el código implementado para el modelo de prioridad de mensajes basado en diccionario (Anexo VII) y el Modelo híbrido para identificación de mensajes prioritarios (Anexo VIII).

Anexo I. Análisis de Polaridad utilizando Clasificación Supervisada para 3 clases: Positivo, Negativo y Neutral

En análisis se emplean dos bases de datos etiquetadas de mensajes de Twitter: La BD Internacional TASS 2018 con 1514 mensajes, y una BD de elaboración propia de 1910 mensajes.

Para ambas bases de datos se generan modelos de clasificación Nayve Bayes y SVN, vectorizando los mensajes con TF y TF-IDF.

Se aplica cross validation para determina qué modelos presentan mejor desempeño y emplearlo en el análisis automático de polaridad de nuevos mensajes en Twitter.

In [2]:
```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from numpy import array
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import NearestCentroid
from sklearn import tree
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn import cross_validation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
C:\Users\lvc\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Deprec
ationWarning: This module was deprecated in version 0.18 in favor of the model_
selection module into which all the refactored classes and functions are moved.
Also note that the interface of the new CV iterators are different from that of
this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Base de Datos TASS

In [2]:
```
#Llamada a BDs TASS

twits_train=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').value
```

In [3]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = twits_train

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sign
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

```
['me', 'caes', 'muy', 'bien', 'tienes', 'que', 'jugar', 'mas', 'partidas', 'a
l', 'lol', 'con', 'russel', 'y', 'conmigo', 'por', 'que', 'tan', 'otako', 'de
ja', 'de', 'ser', 'otako', 'haber', 'si', 'me', 'muero']
['', 'myendlesshazza', 'a', 'que', 'puto', 'mal', 'escribo', 'b', 'me', 'sig
o', 'surrando', 'help', '3', 'ha', 'quedado', 'raro', 'el', 'cometelo', 'ah
i', 'jajajaja']
['', 'estherct209', 'jajajaja', 'la', 'tuya', 'y', 'la', 'd', 'mucha', 'gent
e', 'seguro', 'pero', 'yo', 'no', 'puedo', 'sin', 'mi', 'melena', 'me', 'muer
o']
['quiero', 'mogollon', 'a', 'albabenito99', 'pero', 'sobretodo', 'por', 'lo',
'rapido', 'que', 'contesta', 'a', 'los', 'wasaps']
['vale', 'he', 'visto', 'la', 'tia', 'bebiendose', 'su', 'regla', 'y', 'me',
'hs', 'dado', 'muchs', 'grima']
['', 'yulian_poe', 'guillermoterry1', 'ah', 'mucho', 'mas', 'por', 'supuest
o', 'solo', 'que', 'lo', 'incluyo', 'me', 'habias', 'entendido', 'mal']
['se', 'ha', 'terminado', 'rio2016', 'lamentablemente', 'no', 'arriendo', 'la
```

```
s', 'ganancias', 'al', 'pueblo', 'brasileño', 'por', 'la', 'penuria', 'que',
'les', 'espera', 'suerte', 'y', 'solidaridad']
['11', 'siiii', 'fue', 'super', 'gracioso', 'teniamos', 'que', 'habernos', 's
```

In [11]:

```
#Vectorizado usando TF

vectorizer = CountVectorizer()
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print( vectorizer.vocabulary_ )
```

```
El tamaño del corpus vectorizado es:
(1514, 5904)

Como ejemplo se muestra el vector del primer mensaje:
[[0 0 0 ... 0 0 0]]

El diccionario generado y su frecuencia se muestra a continuación:

{'cae': 802, 'bien': 672, 'jugar': 3011, 'ma': 3340, 'partida': 4116, 'lol':
3280, 'russel': 4878, 'conmigo': 1219, 'tan': 5311, 'otako': 4015, 'deja': 15
15, 'ser': 5016, 'haber': 2487, 'si': 5063, 'muero': 3748, 'myendlesshazza':
3769, 'puto': 4575, 'mal': 3379, 'escribo': 1983, 'sigo': 5077, 'surrando': 5
279, 'help': 2574, 'quedado': 4586, 'raro': 4657, 'cometelo': 1139, 'ahi': 25
2, 'jajajaja': 2882, 'estherct209': 2057, 'mucha': 3743, 'gent': 2368, 'segur
o': 4986, 'puedo': 4548, 'melena': 3543, 'quiero': 4613, 'mogollon': 3661, 'a
lbabenito99': 284, 'sobretodo': 5131, 'rapido': 4651, 'contesta': 1266, 'wasa
p': 5800, 'vale': 5604, 'visto': 5746, 'tia': 5395, 'bebiendos': 634, 'regl
a': 4727, 'hs': 2636, 'dado': 1423, 'much': 3742, 'grima': 2441, 'yulian_po':
5885, 'guillermoterry1': 2466, 'ah': 250, 'supuesto': 5276, 'solo': 5153, 'in
```

In [12]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.44444444 0.49673203 0.49673203 0.46052632 0.61589404 0.50993377
 0.52980132 0.46       0.52666667 0.53333333]
Accuracy: 0.51 (+/- 0.09)
[0.44444444 0.49673203 0.49673203 0.46052632 0.61589404 0.50993377
 0.52980132 0.46       0.52666667 0.53333333]
Precision: 0.51 (+/- 0.09)
[0.44444444 0.49673203 0.49673203 0.46052632 0.61589404 0.50993377
 0.52980132 0.46       0.52666667 0.53333333]
Recall: 0.51 (+/- 0.09)
[0.44444444 0.49673203 0.49673203 0.46052632 0.61589404 0.50993377
 0.52980132 0.46       0.52666667 0.53333333]
F1_Score: 0.51 (+/- 0.09)
```

In [13]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.43137255 0.39869281 0.47058824 0.44078947 0.39735099 0.39735099
 0.31125828 0.38       0.38666667 0.44666667]
Accuracy: 0.41 (+/- 0.08)
[0.43137255 0.39869281 0.47058824 0.44078947 0.39735099 0.39735099
 0.31125828 0.38       0.38666667 0.44666667]
Precision: 0.41 (+/- 0.08)
[0.43137255 0.39869281 0.47058824 0.44078947 0.39735099 0.39735099
 0.31125828 0.38       0.38666667 0.44666667]
Recall: 0.41 (+/- 0.08)
[0.43137255 0.39869281 0.47058824 0.44078947 0.39735099 0.39735099
 0.31125828 0.38       0.38666667 0.44666667]
F1_Score: 0.41 (+/- 0.08)
```

In [14]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.50326797 0.51633987 0.48366013 0.48684211 0.51655629 0.47019868
 0.56953642 0.48       0.50666667 0.52666667]
Accuracy: 0.51 (+/- 0.05)
[0.50326797 0.51633987 0.48366013 0.48684211 0.51655629 0.47019868
 0.56953642 0.48       0.50666667 0.52666667]
Precision: 0.51 (+/- 0.05)
[0.50326797 0.51633987 0.48366013 0.48684211 0.51655629 0.47019868
 0.56953642 0.48       0.50666667 0.52666667]
Recall: 0.51 (+/- 0.05)
[0.50326797 0.51633987 0.48366013 0.48684211 0.51655629 0.47019868
 0.56953642 0.48       0.50666667 0.52666667]
F1_Score: 0.51 (+/- 0.05)
```

In [15]:
```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.43137255 0.49019608 0.50326797 0.44736842 0.50993377 0.47019868
 0.47682119 0.49333333 0.46666667 0.52666667]
Accuracy: 0.48 (+/- 0.06)
[0.40522876 0.49673203 0.45098039 0.44078947 0.57615894 0.45033113
 0.50993377 0.48666667 0.47333333 0.50666667]
Precision: 0.48 (+/- 0.09)
[0.44444444 0.49673203 0.41830065 0.46052632 0.50993377 0.48344371
 0.52980132 0.44       0.46       0.52666667]
Recall: 0.48 (+/- 0.07)
[0.4379085  0.46405229 0.47712418 0.44736842 0.50993377 0.45695364
 0.50993377 0.47333333 0.48       0.52666667]
F1_Score: 0.48 (+/- 0.06)
```

In [16]:
```
#Vectorizado usando TF-IDF

vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(1514, 5904)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'cae': 802, 'bien': 672, 'jugar': 3011, 'ma': 3340, 'partida': 4116, 'lol':
3280, 'russel': 4878, 'conmigo': 1219, 'tan': 5311, 'otako': 4015, 'deja': 15
15, 'ser': 5016, 'haber': 2487, 'si': 5063, 'muero': 3748, 'myendlesshazza':
3769, 'puto': 4575, 'mal': 3379, 'escribo': 1983, 'sigo': 5077, 'surrando': 5
279, 'help': 2574, 'quedado': 4586, 'raro': 4657, 'cometelo': 1139, 'ahi': 25
2, 'jajajaja': 2882, 'estherct209': 2057, 'mucha': 3743, 'gent': 2368, 'segur
o': 4986, 'puedo': 4548, 'melena': 3543, 'quiero': 4613, 'mogollon': 3661, 'a
lbabenito99': 284, 'sobretodo': 5131, 'rapido': 4651, 'contesta': 1266, 'wasa
p': 5800, 'vale': 5604, 'visto': 5746, 'tia': 5395, 'bebiendos': 634, 'regl
a': 4727, 'hs': 2636, 'dado': 1423, 'much': 3742, 'grima': 2441, 'yulian_po':
5885, 'guillermoterry1': 2466, 'ah': 250, 'supuesto': 5276, 'solo': 5153, 'in
```

In [17]:
```
#Cross Validation y exactitud SVM

clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.49019608 0.53594771 0.53594771 0.53289474 0.58940397 0.49006623
 0.56291391 0.51333333 0.50666667 0.58      ]
Accuracy: 0.53 (+/- 0.07)
[0.49019608 0.53594771 0.53594771 0.53289474 0.58940397 0.49006623
 0.56291391 0.51333333 0.50666667 0.58      ]
Precision: 0.53 (+/- 0.07)
[0.49019608 0.53594771 0.53594771 0.53289474 0.58940397 0.49006623
 0.56291391 0.51333333 0.50666667 0.58      ]
Recall: 0.53 (+/- 0.07)
[0.49019608 0.53594771 0.53594771 0.53289474 0.58940397 0.49006623
 0.56291391 0.51333333 0.50666667 0.58      ]
F1_Score: 0.53 (+/- 0.07)
```

In [18]:
```
#Cross Validation y exactitud NB

clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.4379085  0.37254902 0.4379085  0.42763158 0.40397351 0.41721854
 0.31788079 0.36666667 0.4        0.44666667]
Accuracy: 0.40 (+/- 0.08)
[0.4379085  0.37254902 0.4379085  0.42763158 0.40397351 0.41721854
 0.31788079 0.36666667 0.4        0.44666667]
Precision: 0.40 (+/- 0.08)
[0.4379085  0.37254902 0.4379085  0.42763158 0.40397351 0.41721854
 0.31788079 0.36666667 0.4        0.44666667]
Recall: 0.40 (+/- 0.08)
[0.4379085  0.37254902 0.4379085  0.42763158 0.40397351 0.41721854
 0.31788079 0.36666667 0.4        0.44666667]
F1_Score: 0.40 (+/- 0.08)
```

In [19]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.49673203 0.56862745 0.52941176 0.50657895 0.50331126 0.47682119
 0.58940397 0.46       0.52666667 0.54      ]
Accuracy: 0.52 (+/- 0.08)
[0.49673203 0.56862745 0.52941176 0.50657895 0.50331126 0.47682119
 0.58940397 0.46       0.52666667 0.54      ]
Precision: 0.52 (+/- 0.08)
[0.49673203 0.56862745 0.52941176 0.50657895 0.50331126 0.47682119
 0.58940397 0.46       0.52666667 0.54      ]
Recall: 0.52 (+/- 0.08)
[0.49673203 0.56862745 0.52941176 0.50657895 0.50331126 0.47682119
 0.58940397 0.46       0.52666667 0.54      ]
F1_Score: 0.52 (+/- 0.08)
```

In [20]:

```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.48366013 0.46405229 0.47058824 0.47368421 0.49668874 0.47682119
 0.51655629 0.46       0.4        0.52      ]
Accuracy: 0.48 (+/- 0.06)
[0.47712418 0.47712418 0.46405229 0.47368421 0.45033113 0.47682119
 0.52317881 0.44       0.38666667 0.49333333]
Precision: 0.47 (+/- 0.07)
[0.48366013 0.46405229 0.49019608 0.42105263 0.47682119 0.45695364
 0.49668874 0.46666667 0.38       0.48666667]
Recall: 0.46 (+/- 0.07)
[0.47712418 0.47058824 0.47712418 0.45394737 0.49006623 0.46357616
 0.50993377 0.41333333 0.4        0.49333333]
F1_Score: 0.46 (+/- 0.07)
```

Base de datos propia

In [21]:

```
#Llamada a BDs Propia

twits_train=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').text
twits_train_y=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').Polaridad
```

In [22]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = twits_train

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sign
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

```
['', 'vas', 'y', 'chingas', 'a', 'tu', 'madre', 'dedicado', 'a', 'toda', 'l
a', 'prensaprostituta', 'prensacorrupta', 'prensasicaria', 'https', 't', 'c
o', 'kyug3vzef7']
['', 'cencos', 'silviachocarro', 'periodistasapie', 'ifjglobal', 'ifexalc',
'cmdpdh', 'rsf_esp', 'propuestacivica', 'wanifra_latam', 'article19mex', 'glo
balfreemedia', 'apeg_colectivo', 'nos', 'da', 'gusto', 'saver', 'que', 'no',
'solo', 'en', 'mexico', 'existe', 'la', 'prensaprostituta']
['', 'luiscardenasmx', 'ptellovillagran', 'mvstvoficial', 'los', 'pseudoempre
sarios', 'se', 'basaban', 'a', 'invertir', 'por', 'corrupciã³n', 'lo', 'q',
'hace', 'la', 'oposicionmoralmentederrotada', 'es', 'desprestigiar', 'a', 'lo
pezobrador_', 'y', 'la', '4t', 'generando', 'violencia', 'y', 'mã', 's', 'vio
lencia', 'sumando', 'claro', 'con', 'prensac']
['', 'patriciomonero', 'no', 'todo', 'es', 'fanatismo', 'tampoco', 'no', 'exa
geres', 'te', 'exorto', 'a', 'que', 'mires', 'los', 'noticieros', 'de', 'mile
niotv', 'de', 'al', 'medio', 'dia', 'para', 'que', 'te', 'des', 'cuenta', 'co
mo', 'distorcionan', 'las', 'noticias', 'de', 'amlo', 'prensaprostituta']
```

```
['', 'reforma', 'entiendan', 'entre', 'mã', 's', 'lo', 'ataquen', 'mã', 's',
'lo', 'fortalecen', 'pero', 'la', 'prensaprostituta', 'vive', 'en', 'el', 'pa
sado', 'por', 'eso', 'reformatodolodeforma']
```

In [23]:

```
#Vectorizado usando TF

vectorizer = CountVectorizer()
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print( vectorizer.vocabulary_ )
```

```
El tamaño del corpus vectorizado es:
(1911, 10934)

Como ejemplo se muestra el vector del primer mensaje:
[[0 0 0 ... 0 0 0]]

El diccionario generado y su frecuencia se muestra a continuación:

{'va': 10305, 'chinga': 2199, 'madr': 6268, 'dedicado': 3071, 'toda': 9908,
'prensaprostituta': 7977, 'prensacorrupta': 7971, 'prensasicaria': 7978, 'kyu
g3vzef7': 5831, 'cenco': 2086, 'silviachocarro': 9349, 'periodistasapi': 765
3, 'ifjglob': 5149, 'ifexalc': 5148, 'cmdpdh': 2333, 'rsf_esp': 8972, 'propue
stacivica': 8120, 'wanifra_latam': 10628, 'article19mex': 1212, 'globalfreeme
dia': 4712, 'apeg_colectivo': 1057, 'da': 2983, 'gusto': 4857, 'saver': 9140,
'solo': 9436, 'mexico': 6586, 'exist': 4179, 'luiscardenasmx': 6214, 'ptellov
illagran': 8179, 'mvstvofici': 6846, 'pseudoempresario': 8168, 'basaban': 150
0, 'invertir': 5470, 'corrupciã³n': 2767, 'hace': 4905, 'oposicionmoralmented
errotada': 7299, 'desprestigiar': 3288, 'lopezobrador_': 6161, '4t': 269, 'ge
nerando': 4661, 'violencia': 10492, 'mã': 6855, 'sumando': 9580, 'claro': 230
```

In [24]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.57291667 0.47395833 0.56770833 0.5        0.58115183 0.42408377
 0.52356021 0.47643979 0.36315789 0.42857143]
Accuracy: 0.49 (+/- 0.14)
[0.57291667 0.47395833 0.56770833 0.5        0.58115183 0.42408377
 0.52356021 0.47643979 0.36315789 0.42857143]
Precision: 0.49 (+/- 0.14)
[0.57291667 0.47395833 0.56770833 0.5        0.58115183 0.42408377
 0.52356021 0.47643979 0.36315789 0.42857143]
Recall: 0.49 (+/- 0.14)
[0.57291667 0.47395833 0.56770833 0.5        0.58115183 0.42408377
 0.52356021 0.47643979 0.36315789 0.42857143]
F1_Score: 0.49 (+/- 0.14)
```

In [25]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.49479167 0.52083333 0.47916667 0.50520833 0.43979058 0.43979058
 0.48691099 0.42931937 0.34736842 0.46560847]
Accuracy: 0.46 (+/- 0.09)
[0.49479167 0.52083333 0.47916667 0.50520833 0.43979058 0.43979058
 0.48691099 0.42931937 0.34736842 0.46560847]
Precision: 0.46 (+/- 0.09)
[0.49479167 0.52083333 0.47916667 0.50520833 0.43979058 0.43979058
 0.48691099 0.42931937 0.34736842 0.46560847]
Recall: 0.46 (+/- 0.09)
[0.49479167 0.52083333 0.47916667 0.50520833 0.43979058 0.43979058
 0.48691099 0.42931937 0.34736842 0.46560847]
F1_Score: 0.46 (+/- 0.09)
```

In [26]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.52083333 0.4375     0.484375   0.34375    0.4973822  0.46596859
 0.40314136 0.36125654 0.27894737 0.32275132]
Accuracy: 0.41 (+/- 0.16)
[0.52083333 0.4375     0.484375   0.34375    0.4973822  0.46596859
 0.40314136 0.36125654 0.27894737 0.32275132]
Precision: 0.41 (+/- 0.16)
[0.52083333 0.4375     0.484375   0.34375    0.4973822  0.46596859
 0.40314136 0.36125654 0.27894737 0.32275132]
Recall: 0.41 (+/- 0.16)
[0.52083333 0.4375     0.484375   0.34375    0.4973822  0.46596859
 0.40314136 0.36125654 0.27894737 0.32275132]
F1_Score: 0.41 (+/- 0.16)
```

In [27]:

```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.55729167 0.50520833 0.55208333 0.515625   0.4921466  0.43979058
 0.5026178  0.44502618 0.42631579 0.43386243]
Accuracy: 0.49 (+/- 0.09)
[0.53645833 0.52604167 0.57291667 0.52083333 0.48167539 0.41884817
 0.56544503 0.42931937 0.37368421 0.45502646]
Precision: 0.49 (+/- 0.13)
[0.54166667 0.52604167 0.546875   0.53645833 0.5078534  0.42408377
 0.52356021 0.43979058 0.38947368 0.43915344]
Recall: 0.49 (+/- 0.11)
[0.55208333 0.52083333 0.54166667 0.55729167 0.4921466  0.44502618
 0.53926702 0.43979058 0.37368421 0.43386243]
F1_Score: 0.49 (+/- 0.12)
```

In [28]:

```
#Vectorizado usando TF-IDF

vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(1911, 10934)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'va': 10305, 'chinga': 2199, 'madr': 6268, 'dedicado': 3071, 'toda': 9908,
'prensaprostituta': 7977, 'prensacorrupta': 7971, 'prensasicaria': 7978, 'kyu
g3vzef7': 5831, 'cenco': 2086, 'silviachocarro': 9349, 'periodistasapi': 765
3, 'ifjglob': 5149, 'ifexalc': 5148, 'cmdpdh': 2333, 'rsf_esp': 8972, 'propue
stacivica': 8120, 'wanifra_latam': 10628, 'article19mex': 1212, 'globalfreeme
dia': 4712, 'apeg_colectivo': 1057, 'da': 2983, 'gusto': 4857, 'saver': 9140,
'solo': 9436, 'mexico': 6586, 'exist': 4179, 'luiscardenasmx': 6214, 'ptellov
illagran': 8179, 'mvstvofici': 6846, 'pseudoempresario': 8168, 'basaban': 150
0, 'invertir': 5470, 'corrupciã³n': 2767, 'hace': 4905, 'oposicionmoralmented
errotada': 7299, 'desprestigiar': 3288, 'lopezobrador_': 6161, '4t': 269, 'ge
nerando': 4661, 'violencia': 10492, 'mã': 6855, 'sumando': 9580, 'claro': 230
```

In [29]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.609375   0.57291667 0.59895833 0.56770833 0.56020942 0.48691099
 0.56544503 0.41884817 0.40526316 0.44973545]
Accuracy: 0.52 (+/- 0.14)
[0.609375   0.57291667 0.59895833 0.56770833 0.56020942 0.48691099
 0.56544503 0.41884817 0.40526316 0.44973545]
Precision: 0.52 (+/- 0.14)
[0.609375   0.57291667 0.59895833 0.56770833 0.56020942 0.48691099
 0.56544503 0.41884817 0.40526316 0.44973545]
Recall: 0.52 (+/- 0.14)
[0.609375   0.57291667 0.59895833 0.56770833 0.56020942 0.48691099
 0.56544503 0.41884817 0.40526316 0.44973545]
F1_Score: 0.52 (+/- 0.14)
```

In [30]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.46875    0.50520833 0.48958333 0.5        0.45549738 0.44502618
 0.4921466  0.41884817 0.36315789 0.48148148]
Accuracy: 0.46 (+/- 0.08)
[0.46875    0.50520833 0.48958333 0.5        0.45549738 0.44502618
 0.4921466  0.41884817 0.36315789 0.48148148]
Precision: 0.46 (+/- 0.08)
[0.46875    0.50520833 0.48958333 0.5        0.45549738 0.44502618
 0.4921466  0.41884817 0.36315789 0.48148148]
Recall: 0.46 (+/- 0.08)
[0.46875    0.50520833 0.48958333 0.5        0.45549738 0.44502618
 0.4921466  0.41884817 0.36315789 0.48148148]
F1_Score: 0.46 (+/- 0.08)
```

In [31]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.56770833 0.53125    0.59375    0.41666667 0.57068063 0.48167539
 0.5078534  0.40314136 0.35263158 0.38095238]
Accuracy: 0.48 (+/- 0.17)
[0.56770833 0.53125    0.59375    0.41666667 0.57068063 0.48167539
 0.5078534  0.40314136 0.35263158 0.38095238]
Precision: 0.48 (+/- 0.17)
[0.56770833 0.53125    0.59375    0.41666667 0.57068063 0.48167539
 0.5078534  0.40314136 0.35263158 0.38095238]
Recall: 0.48 (+/- 0.17)
[0.56770833 0.53125    0.59375    0.41666667 0.57068063 0.48167539
 0.5078534  0.40314136 0.35263158 0.38095238]
F1_Score: 0.48 (+/- 0.17)
```

In [32]:

```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.484375   0.46354167 0.52083333 0.53645833 0.47120419 0.41884817
 0.53403141 0.42408377 0.38947368 0.43386243]
Accuracy: 0.47 (+/- 0.10)
[0.484375   0.5        0.53125    0.53125    0.5078534  0.45026178
 0.54450262 0.43979058 0.37368421 0.43386243]
Precision: 0.48 (+/- 0.10)
[0.51041667 0.50520833 0.53125    0.515625   0.46596859 0.42931937
 0.52879581 0.42931937 0.34210526 0.3968254 ]
Recall: 0.47 (+/- 0.12)
[0.53125    0.53645833 0.53645833 0.55729167 0.51832461 0.43979058
 0.53926702 0.43979058 0.37894737 0.4021164 ]
F1_Score: 0.49 (+/- 0.12)
```

Análisis cruzado de sentimientos usando ambasbases de datos

Clasificador entrenado con TASS y probado con BD-Propia

In [21]:

```
#Llamando archivos

twits_train=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').Polaridad

datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

In [22]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = ttrain

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    #print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sig
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [23]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(3425, 15229)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'cae': 2401, 'bien': 2071, 'jugar': 7914, 'ma': 8713, 'partida': 10539, 'lo
l': 8549, 'russel': 12549, 'conmigo': 3486, 'tan': 13557, 'otako': 10323, 'de
ja': 4192, 'ser': 12920, 'haber': 6675, 'si': 13001, 'muero': 9571, 'myendles
shazza': 9629, 'puto': 11576, 'mal': 8775, 'escribo': 5411, 'sigo': 13032, 's
urrando': 13468, 'help': 6820, 'quedado': 11651, 'raro': 11840, 'cometelo': 3
280, 'ahi': 909, 'jajajaja': 7657, 'estherct209': 5554, 'mucha': 9560, 'gen
t': 6368, 'seguro': 12830, 'puedo': 11521, 'melena': 9133, 'quiero': 11711,
'mogollon': 9403, 'albabenito99': 974, 'sobretodo': 13144, 'rapido': 11829,
'contesta': 3597, 'wasap': 14829, 'vale': 14392, 'visto': 14679, 'tia': 1377
7, 'bebiendos': 1993, 'regla': 12096, 'hs': 6970, 'dado': 4021, 'much': 9559,
'grima': 6563, 'yulian_po': 15093, 'guillermoterry1': 6622, 'ah': 904, 'supue
```

In [24]:

```
train=features[0:(len(twits_train)-1)]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
pred=clsfcdr.predict(train)
print("Vector de clases resultante para cada uno de los mensajes del conjunto
print(pred)
print ("La exactitud del clasificador con el conjunto de entrenamiento es = %
```

```
Vector de clases resultante para cada uno de los mensajes del conjunto de entre
namiento:
[ 0 -1 -1 ... -1 -1  0]
La exactitud del clasificador con el conjunto de entrenamiento es = 0.976867151
354924
```

In [25]:
```
test = features[len(twits_train):len(ttrain)]
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.35374149659863946
Precision del clasificador con el conjunto de prueba es = 0.35374149659863946
Recall del clasificador con el conjunto de prueba es = 0.35374149659863946
F1 del clasificador con el conjunto de prueba es = 0.35374149659863946
```

In [39]:
```
clsfcdr=GaussianNB()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.35583464154892724
Precision del clasificador con el conjunto de prueba es = 0.35583464154892724
Recall del clasificador con el conjunto de prueba es = 0.35583464154892724
F1 del clasificador con el conjunto de prueba es = 0.3558346415489272
```

In [41]:
```
clsfcdr=NearestCentroid()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.39089481946624804
Precision del clasificador con el conjunto de prueba es = 0.39089481946624804
Recall del clasificador con el conjunto de prueba es = 0.39089481946624804
F1 del clasificador con el conjunto de prueba es = 0.390894819466248
```

In [42]:
```
clsfcdr=tree.DecisionTreeClassifier()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.3500784929356358
Precision del clasificador con el conjunto de prueba es = 0.3500784929356358
Recall del clasificador con el conjunto de prueba es = 0.3500784929356358
F1 del clasificador con el conjunto de prueba es = 0.3500784929356358
```

Clasificador entrenado con BD-Propia y probadoconTASS

In [8]:
```
train=features[len(twits_train):len(ttrain)]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,twits_test_y)
pred=clsfcdr.predict(train)
print("Vector de clases resultante para cada uno de los mensajes del conjunto
print(pred)
print ("La exactitud del clasificador con el conjunto de entrenamiento es = %
```

```
Vector de clases resultante para cada uno de los mensajes del conjunto de entre
namiento:
[-1  1  1 ...  0  1 -1]
La exactitud del clasificador con el conjunto de entrenamiento es = 0.922553636
8393511
```

In [9]:
```
test = features[0:len(twits_train)]
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.2965653896961691
Precision del clasificador con el conjunto de prueba es = 0.2965653896961691
Recall del clasificador con el conjunto de prueba es = 0.2965653896961691
F1 del clasificador con el conjunto de prueba es = 0.2965653896961691
```

In [12]:
```
clsfcdr=GaussianNB()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.3797886393659181
Precision del clasificador con el conjunto de prueba es = 0.3797886393659181
Recall del clasificador con el conjunto de prueba es = 0.3797886393659181
F1 del clasificador con el conjunto de prueba es = 0.3797886393659181
```

In [13]:
```
clsfcdr=NearestCentroid()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.2952443857331572
Precision del clasificador con el conjunto de prueba es = 0.2952443857331572
Recall del clasificador con el conjunto de prueba es = 0.2952443857331572
F1 del clasificador con el conjunto de prueba es = 0.2952443857331572
```

In [14]:
```
clsfcdr=tree.DecisionTreeClassifier()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.31836195508586523
Precision del clasificador con el conjunto de prueba es = 0.31836195508586523
Recall del clasificador con el conjunto de prueba es = 0.31836195508586523
F1 del clasificador con el conjunto de prueba es = 0.31836195508586523
```

Validación cruzada con ambas bases de datos

In [10]:
```
#Concatenando vectores de etiquetas

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

In [11]:

```
#Cross Validation y exactitud SVM
clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador SVM")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud NB
clf = GaussianNB()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador Naive Bayes")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud NearestCentroid
clf = NearestCentroid()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador Nearest Centroid")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud DecisionTrees
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador DecisionTrees")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
```

```
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Clasificador SVM
[0.51453488 0.47093023 0.48104956 0.55685131 0.57434402 0.62280702
 0.55263158 0.50877193 0.39296188 0.36656891]
Accuracy: 0.50 (+/- 0.15)
[0.51453488 0.47093023 0.48104956 0.55685131 0.57434402 0.62280702
 0.55263158 0.50877193 0.39296188 0.36656891]
Precision: 0.50 (+/- 0.15)
[0.51453488 0.47093023 0.48104956 0.55685131 0.57434402 0.62280702
 0.55263158 0.50877193 0.39296188 0.36656891]
Recall: 0.50 (+/- 0.15)
[0.51453488 0.47093023 0.48104956 0.55685131 0.57434402 0.62280702
 0.55263158 0.50877193 0.39296188 0.36656891]
F1_Score: 0.50 (+/- 0.15)
Clasificador Naive Bayes
[0.40116279 0.36337209 0.3819242  0.44314869 0.46355685 0.5380117
 0.47660819 0.44736842 0.37536657 0.38416422]
Accuracy: 0.43 (+/- 0.11)
[0.40116279 0.36337209 0.3819242  0.44314869 0.46355685 0.5380117
 0.47660819 0.44736842 0.37536657 0.38416422]
Precision: 0.43 (+/- 0.11)
[0.40116279 0.36337209 0.3819242  0.44314869 0.46355685 0.5380117
 0.47660819 0.44736842 0.37536657 0.38416422]
Recall: 0.43 (+/- 0.11)
[0.40116279 0.36337209 0.3819242  0.44314869 0.46355685 0.5380117
 0.47660819 0.44736842 0.37536657 0.38416422]
F1_Score: 0.43 (+/- 0.11)
Clasificador Nearest Centroid
[0.4622093  0.46511628 0.49271137 0.58892128 0.58017493 0.61988304
 0.53216374 0.52631579 0.36363636 0.34897361]
Accuracy: 0.50 (+/- 0.17)
[0.4622093  0.46511628 0.49271137 0.58892128 0.58017493 0.61988304
 0.53216374 0.52631579 0.36363636 0.34897361]
Precision: 0.50 (+/- 0.17)
[0.4622093  0.46511628 0.49271137 0.58892128 0.58017493 0.61988304
 0.53216374 0.52631579 0.36363636 0.34897361]
Recall: 0.50 (+/- 0.17)
[0.4622093  0.46511628 0.49271137 0.58892128 0.58017493 0.61988304
 0.53216374 0.52631579 0.36363636 0.34897361]
F1_Score: 0.50 (+/- 0.17)
Clasificador DecisionTrees
[0.44767442 0.43023256 0.41107872 0.45772595 0.49562682 0.50877193
 0.48830409 0.4122807  0.4398827  0.37243402]
Accuracy: 0.45 (+/- 0.08)
[0.4244186  0.40697674 0.41982507 0.48104956 0.49271137 0.51754386
 0.46783626 0.42690058 0.41935484 0.36950147]
Precision: 0.44 (+/- 0.09)
[0.4505814  0.40697674 0.40233236 0.47230321 0.48688047 0.50584795
 0.5        0.41520468 0.42815249 0.38416422]
Recall: 0.45 (+/- 0.08)
[0.43313953 0.40988372 0.42274052 0.44314869 0.48979592 0.53216374
```

```
 0.49707602 0.38888889 0.43695015 0.36363636]
F1_Score: 0.44 (+/- 0.10)
```

In [ ]:

```
1
```

Anexo II. Análisis de Polaridad utilizando Clasificación Supervisada para 2 clases: Positivo y Negativo

En análisis se emplean dos bases de datos etiquetadas de mensajes de Twitter: La BD Internacional TASS 2018 con 1111 mensajes, y una BD de elaboración propia de 1022 mensajes.

Para ambas bases de datos se generan modelos de clasificación SVM, Nayve Bayes, NearestCentroid y Decision Tree, vectorizando los mensajes con TF y TF-IDF.

Se aplica cross validation para determina qué modelos presentan mejor desempeño y emplearlo en el análisis automático de polaridad de nuevos mensajes en Twitter.

In [1]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from numpy import array
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import NearestCentroid
from sklearn import tree
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn import cross_validation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score

from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
C:\Users\lvc\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Deprec
ationWarning: This module was deprecated in version 0.18 in favor of the model_
selection module into which all the refactored classes and functions are moved.
Also note that the interface of the new CV iterators are different from that of
this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Base de Datos TASS

In [3]:

```
#Llamada a BDs TASS

twits_train=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').value
```

In [4]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = twits_train

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    #print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sig
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [5]:

```
#Vectorizado usando TF

vectorizer = CountVectorizer()
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print( vectorizer.vocabulary_ )
```

```
El tamaño del corpus vectorizado es:
(1111, 4638)

Como ejemplo se muestra el vector del primer mensaje:
[[0 0 0 ... 0 0 0]]

El diccionario generado y su frecuencia se muestra a continuación:

{'myendlesshazza': 2979, 'puto': 3603, 'mal': 2670, 'escribo': 1572, 'sigo':
3996, 'surrando': 4158, 'help': 2056, 'quedado': 3610, 'raro': 3667, 'cometel
o': 904, 'ahi': 194, 'jajajaja': 2291, 'estherct209': 1634, 'mucha': 2957, 'g
ent': 1881, 'seguro': 3921, 'puedo': 3584, 'melena': 2798, 'muero': 2961, 'va
le': 4404, 'visto': 4524, 'tia': 4241, 'bebiendos': 503, 'regla': 3722, 'hs':
2104, 'dado': 1130, 'much': 2956, 'grima': 1946, 'terminado': 4225, 'rio201
6': 3787, 'lamentablement': 2461, 'arriendo': 376, 'ganancia': 1854, 'puebl
o': 3580, 'brasileño': 582, 'penuria': 3316, 'espera': 1603, 'suert': 4136,
'solidaridad': 4052, 'toni_end': 4288, 'seria': 3955, 'mejor': 2794, 'dejase
n': 1208, 'emitir': 1458, 'basura': 486, 'evolucionar': 1659, 'bien': 535, 'j
onoro96': 2351, 'mandaria': 2694, 'comprart': 943, 'burro': 615, 'creo': 106
```

In [6]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.69642857 0.69642857 0.71428571 0.61607143 0.81081081 0.72072072
 0.7027027  0.69090909 0.65454545 0.78181818]
Accuracy: 0.71 (+/- 0.11)
[0.69642857 0.69642857 0.71428571 0.61607143 0.81081081 0.72072072
 0.7027027  0.69090909 0.65454545 0.78181818]
Precision: 0.71 (+/- 0.11)
[0.69642857 0.69642857 0.71428571 0.61607143 0.81081081 0.72072072
 0.7027027  0.69090909 0.65454545 0.78181818]
Recall: 0.71 (+/- 0.11)
[0.69642857 0.69642857 0.71428571 0.61607143 0.81081081 0.72072072
 0.7027027  0.69090909 0.65454545 0.78181818]
F1_Score: 0.71 (+/- 0.11)
```

In [7]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.65178571 0.67857143 0.64285714 0.625      0.59459459 0.56756757
 0.57657658 0.50909091 0.6        0.64545455]
Accuracy: 0.61 (+/- 0.09)
[0.65178571 0.67857143 0.64285714 0.625      0.59459459 0.56756757
 0.57657658 0.50909091 0.6        0.64545455]
Precision: 0.61 (+/- 0.09)
[0.65178571 0.67857143 0.64285714 0.625      0.59459459 0.56756757
 0.57657658 0.50909091 0.6        0.64545455]
Recall: 0.61 (+/- 0.09)
[0.65178571 0.67857143 0.64285714 0.625      0.59459459 0.56756757
 0.57657658 0.50909091 0.6        0.64545455]
F1_Score: 0.61 (+/- 0.09)
```

In [8]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.69642857 0.69642857 0.66964286 0.64285714 0.68468468 0.67567568
 0.72072072 0.70909091 0.65454545 0.72727273]
Accuracy: 0.69 (+/- 0.05)
[0.69642857 0.69642857 0.66964286 0.64285714 0.68468468 0.67567568
 0.72072072 0.70909091 0.65454545 0.72727273]
Precision: 0.69 (+/- 0.05)
[0.69642857 0.69642857 0.66964286 0.64285714 0.68468468 0.67567568
 0.72072072 0.70909091 0.65454545 0.72727273]
Recall: 0.69 (+/- 0.05)
[0.69642857 0.69642857 0.66964286 0.64285714 0.68468468 0.67567568
 0.72072072 0.70909091 0.65454545 0.72727273]
F1_Score: 0.69 (+/- 0.05)
```

In [9]:

```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.66071429 0.6875     0.66071429 0.65178571 0.72072072 0.67567568
 0.67567568 0.62727273 0.65454545 0.75454545]
Accuracy: 0.68 (+/- 0.07)
[0.64285714 0.70535714 0.64285714 0.64285714 0.68468468 0.69369369
 0.72072072 0.62727273 0.60909091 0.74545455]
Precision: 0.67 (+/- 0.08)
[0.66964286 0.66964286 0.66964286 0.65178571 0.69369369 0.67567568
 0.71171171 0.67272727 0.62727273 0.77272727]
Recall: 0.68 (+/- 0.07)
[0.64285714 0.66071429 0.64285714 0.63392857 0.68468468 0.62162162
 0.68468468 0.65454545 0.62727273 0.74545455]
F1_Score: 0.66 (+/- 0.07)
```

In [10]:

```
#Vectorizado usando TF-IDF

vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(1111, 4638)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'myendlesshazza': 2979, 'puto': 3603, 'mal': 2670, 'escribo': 1572, 'sigo':
3996, 'surrando': 4158, 'help': 2056, 'quedado': 3610, 'raro': 3667, 'cometel
o': 904, 'ahi': 194, 'jajajaja': 2291, 'estherct209': 1634, 'mucha': 2957, 'g
ent': 1881, 'seguro': 3921, 'puedo': 3584, 'melena': 2798, 'muero': 2961, 'va
le': 4404, 'visto': 4524, 'tia': 4241, 'bebiendos': 503, 'regla': 3722, 'hs':
2104, 'dado': 1130, 'much': 2956, 'grima': 1946, 'terminado': 4225, 'rio201
6': 3787, 'lamentablement': 2461, 'arriendo': 376, 'ganancia': 1854, 'puebl
o': 3580, 'brasileño': 582, 'penuria': 3316, 'espera': 1603, 'suert': 4136,
'solidaridad': 4052, 'toni_end': 4288, 'seria': 3955, 'mejor': 2794, 'dejase
n': 1208, 'emitir': 1458, 'basura': 486, 'evolucionar': 1659, 'bien': 535, 'j
onoro96': 2351, 'mandaria': 2694, 'comprart': 943, 'burro': 615, 'creo': 106
```

In [11]:

```
#Cross Validation y exactitud SVM

clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.67857143 0.73214286 0.75       0.69642857 0.75675676 0.72972973
 0.73873874 0.72727273 0.70909091 0.74545455]
Accuracy: 0.73 (+/- 0.05)
[0.67857143 0.73214286 0.75       0.69642857 0.75675676 0.72972973
 0.73873874 0.72727273 0.70909091 0.74545455]
Precision: 0.73 (+/- 0.05)
[0.67857143 0.73214286 0.75       0.69642857 0.75675676 0.72972973
 0.73873874 0.72727273 0.70909091 0.74545455]
Recall: 0.73 (+/- 0.05)
[0.67857143 0.73214286 0.75       0.69642857 0.75675676 0.72972973
 0.73873874 0.72727273 0.70909091 0.74545455]
F1_Score: 0.73 (+/- 0.05)
```

In [12]:

```
#Cross Validation y exactitud NB

clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.66964286 0.66071429 0.61607143 0.625      0.59459459 0.59459459
 0.57657658 0.50909091 0.59090909 0.64545455]
Accuracy: 0.61 (+/- 0.09)
[0.66964286 0.66071429 0.61607143 0.625      0.59459459 0.59459459
 0.57657658 0.50909091 0.59090909 0.64545455]
Precision: 0.61 (+/- 0.09)
[0.66964286 0.66071429 0.61607143 0.625      0.59459459 0.59459459
 0.57657658 0.50909091 0.59090909 0.64545455]
Recall: 0.61 (+/- 0.09)
[0.66964286 0.66071429 0.61607143 0.625      0.59459459 0.59459459
 0.57657658 0.50909091 0.59090909 0.64545455]
F1_Score: 0.61 (+/- 0.09)
```

In [13]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.70535714 0.72321429 0.73214286 0.70535714 0.76576577 0.7027027
 0.73873874 0.70909091 0.7        0.74545455]
Accuracy: 0.72 (+/- 0.04)
[0.70535714 0.72321429 0.73214286 0.70535714 0.76576577 0.7027027
 0.73873874 0.70909091 0.7        0.74545455]
Precision: 0.72 (+/- 0.04)
[0.70535714 0.72321429 0.73214286 0.70535714 0.76576577 0.7027027
 0.73873874 0.70909091 0.7        0.74545455]
Recall: 0.72 (+/- 0.04)
[0.70535714 0.72321429 0.73214286 0.70535714 0.76576577 0.7027027
 0.73873874 0.70909091 0.7        0.74545455]
F1_Score: 0.72 (+/- 0.04)
```

In [14]:
```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.625      0.64285714 0.66071429 0.60714286 0.69369369 0.64864865
 0.63063063 0.60909091 0.64545455 0.68181818]
Accuracy: 0.64 (+/- 0.05)
[0.63392857 0.61607143 0.65178571 0.625      0.67567568 0.62162162
 0.62162162 0.62727273 0.60909091 0.69090909]
Precision: 0.64 (+/- 0.05)
[0.64285714 0.61607143 0.70535714 0.61607143 0.65765766 0.62162162
 0.57657658 0.60909091 0.7        0.69090909]
Recall: 0.64 (+/- 0.08)
[0.625      0.67857143 0.66071429 0.60714286 0.67567568 0.63963964
 0.57657658 0.60909091 0.65454545 0.68181818]
F1_Score: 0.64 (+/- 0.07)
```

Base de datos propia

In [15]:
```
#Llamada a BDs Propia

twits_train=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').text
twits_train_y=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').Polaridad
```

In [16]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = twits_train

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    #print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sig
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [17]:

```
#Vectorizado usando TF

vectorizer = CountVectorizer()
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print( vectorizer.vocabulary_ )
```

```
El tamaño del corpus vectorizado es:
(1022, 6862)

Como ejemplo se muestra el vector del primer mensaje:
[[0 0 0 ... 0 0 0]]

El diccionario generado y su frecuencia se muestra a continuación:

{'va': 6484, 'chinga': 1349, 'madr': 3921, 'dedicado': 1907, 'toda': 6240, 'p
rensaprostituta': 5027, 'prensacorrupta': 5022, 'prensasicaria': 5028, 'kyug3
vzef7': 3660, 'fakenew': 2672, 'lugar': 3891, 'noticia': 4446, 'decimo': 189
3, 'p2r9is1ktz': 4639, 'bebeka04': 923, 'uptothemoth': 6450, 'oposicionmoralm
entederrotada': 4584, 'cree': 1745, 'falsa': 2678, 'difund': 2129, 'pendejiz
a': 4784, 'prian': 5071, 'indigna': 3304, 'mismo': 4196, 'provocaron': 5153,
'carlosloret': 1203, 'escrib': 2503, 'pura': 5203, 'basura': 912, 'estimado':
2561, 'jorgeramosnew': 3563, 'lopezobrador_': 3858, 'vendido': 6523, 'mierd
a': 4161, 'prensachillona': 5020, 'jrisco': 3572, 'periodista': 4806, 'medioc
r': 4076, 'nivel': 4411, 'si': 5866, 'digno': 2138, 'representant': 5508, 'al
ia': 489, 'capulina': 1181, 'afuprjysaw': 391, 'liderfisc': 3776, 'cirogomez
```

In [18]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.72815534 0.77669903 0.75728155 0.78640777 0.69607843 0.7254902
 0.87254902 0.62745098 0.74257426 0.75247525]
Accuracy: 0.75 (+/- 0.12)
[0.72815534 0.77669903 0.75728155 0.78640777 0.69607843 0.7254902
 0.87254902 0.62745098 0.74257426 0.75247525]
Precision: 0.75 (+/- 0.12)
[0.72815534 0.77669903 0.75728155 0.78640777 0.69607843 0.7254902
 0.87254902 0.62745098 0.74257426 0.75247525]
Recall: 0.75 (+/- 0.12)
[0.72815534 0.77669903 0.75728155 0.78640777 0.69607843 0.7254902
 0.87254902 0.62745098 0.74257426 0.75247525]
F1_Score: 0.75 (+/- 0.12)
```

In [19]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.75728155 0.76699029 0.73786408 0.67961165 0.70588235 0.70588235
 0.84313725 0.6372549  0.74257426 0.8019802 ]
Accuracy: 0.74 (+/- 0.11)
[0.75728155 0.76699029 0.73786408 0.67961165 0.70588235 0.70588235
 0.84313725 0.6372549  0.74257426 0.8019802 ]
Precision: 0.74 (+/- 0.11)
[0.75728155 0.76699029 0.73786408 0.67961165 0.70588235 0.70588235
 0.84313725 0.6372549  0.74257426 0.8019802 ]
Recall: 0.74 (+/- 0.11)
[0.75728155 0.76699029 0.73786408 0.67961165 0.70588235 0.70588235
 0.84313725 0.6372549  0.74257426 0.8019802 ]
F1_Score: 0.74 (+/- 0.11)
```

In [20]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.7184466  0.7961165  0.6407767  0.76699029 0.55882353 0.85294118
 0.90196078 0.2745098  0.7029703  0.65346535]
Accuracy: 0.69 (+/- 0.34)
[0.7184466  0.7961165  0.6407767  0.76699029 0.55882353 0.85294118
 0.90196078 0.2745098  0.7029703  0.65346535]
Precision: 0.69 (+/- 0.34)
[0.7184466  0.7961165  0.6407767  0.76699029 0.55882353 0.85294118
 0.90196078 0.2745098  0.7029703  0.65346535]
Recall: 0.69 (+/- 0.34)
[0.7184466  0.7961165  0.6407767  0.76699029 0.55882353 0.85294118
 0.90196078 0.2745098  0.7029703  0.65346535]
F1_Score: 0.69 (+/- 0.34)
```

In [21]:

```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.72815534 0.77669903 0.74757282 0.74757282 0.65686275 0.7745098
 0.87254902 0.5        0.6039604  0.73267327]
Accuracy: 0.71 (+/- 0.20)
[0.70873786 0.7961165  0.72815534 0.72815534 0.66666667 0.73529412
 0.83333333 0.53921569 0.66336634 0.74257426]
Precision: 0.71 (+/- 0.15)
[0.73786408 0.76699029 0.74757282 0.75728155 0.69607843 0.71568627
 0.87254902 0.53921569 0.65346535 0.75247525]
Recall: 0.72 (+/- 0.16)
[0.72815534 0.75728155 0.73786408 0.73786408 0.70588235 0.7745098
 0.85294118 0.51960784 0.62376238 0.74257426]
F1_Score: 0.72 (+/- 0.17)
```

In [22]:

```
#Vectorizado usando TF-IDF

vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(1022, 6862)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'va': 6484, 'chinga': 1349, 'madr': 3921, 'dedicado': 1907, 'toda': 6240, 'p
rensaprostituta': 5027, 'prensacorrupta': 5022, 'prensasicaria': 5028, 'kyug3
vzef7': 3660, 'fakenew': 2672, 'lugar': 3891, 'noticia': 4446, 'decimo': 189
3, 'p2r9is1ktz': 4639, 'bebeka04': 923, 'uptothemoth': 6450, 'oposicionmoralm
entederrotada': 4584, 'cree': 1745, 'falsa': 2678, 'difund': 2129, 'pendejiz
a': 4784, 'prian': 5071, 'indigna': 3304, 'mismo': 4196, 'provocaron': 5153,
'carlosloret': 1203, 'escrib': 2503, 'pura': 5203, 'basura': 912, 'estimado':
2561, 'jorgeramosnew': 3563, 'lopezobrador_': 3858, 'vendido': 6523, 'mierd
a': 4161, 'prensachillona': 5020, 'jrisco': 3572, 'periodista': 4806, 'medioc
r': 4076, 'nivel': 4411, 'si': 5866, 'digno': 2138, 'representant': 5508, 'al
ia': 489, 'capulina': 1181, 'afuprjysaw': 391, 'liderfisc': 3776, 'cirogomez
```

In [23]:

```
#Cross Validation y exactitud SVM


clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.80582524 0.86407767 0.78640777 0.78640777 0.7254902  0.81372549
 0.87254902 0.60784314 0.72277228 0.77227723]
Accuracy: 0.78 (+/- 0.15)
[0.80582524 0.86407767 0.78640777 0.78640777 0.7254902  0.81372549
 0.87254902 0.60784314 0.72277228 0.77227723]
Precision: 0.78 (+/- 0.15)
[0.80582524 0.86407767 0.78640777 0.78640777 0.7254902  0.81372549
 0.87254902 0.60784314 0.72277228 0.77227723]
Recall: 0.78 (+/- 0.15)
[0.80582524 0.86407767 0.78640777 0.78640777 0.7254902  0.81372549
 0.87254902 0.60784314 0.72277228 0.77227723]
F1_Score: 0.78 (+/- 0.15)
```

In [24]:

```
#Cross Validation y exactitud NB


clf = GaussianNB()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.74757282 0.76699029 0.72815534 0.68932039 0.71568627 0.70588235
 0.85294118 0.62745098 0.71287129 0.79207921]
Accuracy: 0.73 (+/- 0.12)
[0.74757282 0.76699029 0.72815534 0.68932039 0.71568627 0.70588235
 0.85294118 0.62745098 0.71287129 0.79207921]
Precision: 0.73 (+/- 0.12)
[0.74757282 0.76699029 0.72815534 0.68932039 0.71568627 0.70588235
 0.85294118 0.62745098 0.71287129 0.79207921]
Recall: 0.73 (+/- 0.12)
[0.74757282 0.76699029 0.72815534 0.68932039 0.71568627 0.70588235
 0.85294118 0.62745098 0.71287129 0.79207921]
F1_Score: 0.73 (+/- 0.12)
```

In [25]:

```
#Cross Validation y exactitud NearestCentroid

clf = NearestCentroid()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.72815534 0.83495146 0.72815534 0.78640777 0.6372549  0.79411765
 0.95098039 0.38235294 0.72277228 0.73267327]
Accuracy: 0.73 (+/- 0.28)
[0.72815534 0.83495146 0.72815534 0.78640777 0.6372549  0.79411765
 0.95098039 0.38235294 0.72277228 0.73267327]
Precision: 0.73 (+/- 0.28)
[0.72815534 0.83495146 0.72815534 0.78640777 0.6372549  0.79411765
 0.95098039 0.38235294 0.72277228 0.73267327]
Recall: 0.73 (+/- 0.28)
[0.72815534 0.83495146 0.72815534 0.78640777 0.6372549  0.79411765
 0.95098039 0.38235294 0.72277228 0.73267327]
F1_Score: 0.73 (+/- 0.28)
```

In [26]:
```
#Cross Validation y exactitud DecisionTrees

clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, twits_train_y, cv=10)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='precis
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='recall
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, twits_train_y, cv=10, scoring='f1_mic
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.73786408 0.66990291 0.70873786 0.74757282 0.60784314 0.76470588
 0.80392157 0.57843137 0.67326733 0.64356436]
Accuracy: 0.69 (+/- 0.14)
[0.72815534 0.68932039 0.68932039 0.72815534 0.57843137 0.76470588
 0.80392157 0.58823529 0.69306931 0.68316832]
Precision: 0.69 (+/- 0.13)
[0.7184466  0.66990291 0.72815534 0.76699029 0.6372549  0.74509804
 0.79411765 0.56862745 0.71287129 0.67326733]
Recall: 0.70 (+/- 0.13)
[0.72815534 0.6407767  0.7184466  0.74757282 0.62745098 0.71568627
 0.75490196 0.52941176 0.7029703  0.65346535]
F1_Score: 0.68 (+/- 0.13)
```

Análisis cruzado de sentimientos usando ambas bases de datos

Clasificador entrenado con TASS y probado con BD-Propia

In [11]:
```
#Llamando archivos

twits_train=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').Polaridad

datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

In [12]:

```
#Limpiando paralabras para construir duccionario
#Eliminado acentos

datext = ttrain

a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    #print (toks) #Se imprimen los tokens de cada twit habiendo eliminado sig
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [13]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:")
print(features.shape)
print("")
print("Como ejemplo se muestra el vector del primer mensaje:")
print(features[0])
print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es:
(2133, 10420)

Como ejemplo se muestra el vector del primer mensaje:
[0. 0. 0. ... 0. 0. 0.]

El diccionario generado y su frecuencia se muestra a continuación:

{'myendlesshazza': 6622, 'puto': 7980, 'mal': 6015, 'escribo': 3717, 'sigo':
8957, 'surrando': 9261, 'help': 4688, 'quedado': 8018, 'raro': 8149, 'cometel
o': 2207, 'ahi': 565, 'jajajaja': 5275, 'estherct209': 3815, 'mucha': 6578,
'gent': 4360, 'seguro': 8811, 'puedo': 7943, 'melena': 6269, 'muero': 6584,
'vale': 9876, 'visto': 10086, 'tia': 9469, 'bebiendos': 1307, 'regla': 8324,
'hs': 4789, 'dado': 2729, 'much': 6577, 'grima': 4501, 'terminado': 9426, 'ri
o2016': 8517, 'lamentablement': 5602, 'arriendo': 1013, 'ganancia': 4298, 'pu
eblo': 7938, 'brasileño': 1467, 'penuria': 7365, 'espera': 3764, 'suert': 920
9, 'solidaridad': 9059, 'toni_end': 9564, 'seria': 8886, 'mejor': 6264, 'deja
sen': 2858, 'emitir': 3486, 'basura': 1280, 'evolucionar': 3874, 'bien': 136
1, 'jonoro96': 5386, 'mandaria': 6051, 'comprart': 2275, 'burro': 1532, 'cre
```

In [14]:

```
train=features[0:(len(twits_train)-1)]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
pred=clsfcdr.predict(train)
print("Vector de clases resultante para cada uno de los mensajes del conjunto
print(pred)
print ("Exactitud del clasificador con el conjunto de entrenamiento es = %s"
```

```
Vector de clases resultante para cada uno de los mensajes del conjunto de entre
namiento:
[-1 -1 -1 ...  1  1  1]
Exactitud del clasificador con el conjunto de entrenamiento es = 0.986486486486
4865
```

In [15]:
```
test = features[len(twits_train):len(ttrain)]
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.6281800391389433
Precision del clasificador con el conjunto de prueba es = 0.6281800391389433
Recall del clasificador con el conjunto de prueba es = 0.6281800391389433
F1 del clasificador con el conjunto de prueba es = 0.6281800391389433
```

In [17]:
```
clsfcdr=GaussianNB()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.5518590998043053
Precision del clasificador con el conjunto de prueba es = 0.5518590998043053
Recall del clasificador con el conjunto de prueba es = 0.5518590998043053
F1 del clasificador con el conjunto de prueba es = 0.5518590998043053
```

In [19]:
```
clsfcdr=NearestCentroid()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.636986301369863
Precision del clasificador con el conjunto de prueba es = 0.636986301369863
Recall del clasificador con el conjunto de prueba es = 0.636986301369863
F1 del clasificador con el conjunto de prueba es = 0.636986301369863
```

In [20]:
```
clsfcdr=tree.DecisionTreeClassifier()
clsfcdr.fit(train,twits_train_y[0:(len(twits_train)-1)])
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.5968688845401174
Precision del clasificador con el conjunto de prueba es = 0.5968688845401174
Recall del clasificador con el conjunto de prueba es = 0.5968688845401174
F1 del clasificador con el conjunto de prueba es = 0.5968688845401174
```

Clasificador entrenado con BD-Propia y probado conTASS

In [21]:
```
train=features[len(twits_train):len(ttrain)]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,twits_test_y)
pred=clsfcdr.predict(train)
print("Vector de clases resultante para cada uno de los mensajes del conjunto
print(pred)
print ("La exactitud del clasificador con el conjunto de entrenamiento es = %
```

```
Vector de clases resultante para cada uno de los mensajes del conjunto de entre
namiento:
[-1 -1 -1 ...  1  1  1]
La exactitud del clasificador con el conjunto de entrenamiento es = 0.981409001
9569471
```

In [22]:
```
test = features[0:len(twits_train)]
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.6111611161116112
Precision del clasificador con el conjunto de prueba es = 0.6111611161116112
Recall del clasificador con el conjunto de prueba es = 0.6111611161116112
F1 del clasificador con el conjunto de prueba es = 0.6111611161116112
```

In [23]:
```
clsfcdr=GaussianNB()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.5724572457245725
Precision del clasificador con el conjunto de prueba es = 0.5724572457245725
Recall del clasificador con el conjunto de prueba es = 0.5724572457245725
F1 del clasificador con el conjunto de prueba es = 0.5724572457245725
```

In [24]:
```
clsfcdr=NearestCentroid()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.6273627362736274
Precision del clasificador con el conjunto de prueba es = 0.6273627362736274
Recall del clasificador con el conjunto de prueba es = 0.6273627362736274
F1 del clasificador con el conjunto de prueba es = 0.6273627362736274
```

In [25]:
```
clsfcdr=tree.DecisionTreeClassifier()
clsfcdr.fit(train,twits_test_y)
predtest=clsfcdr.predict(test)
print ("Exactitud del clasificador con el conjunto de prueba es = %s" % accur
print ("Precision del clasificador con el conjunto de prueba es = %s" % preci
print ("Recall del clasificador con el conjunto de prueba es = %s" % recall_s
print ("F1 del clasificador con el conjunto de prueba es = %s" % f1_score(twi
```

```
Exactitud del clasificador con el conjunto de prueba es = 0.5904590459045904
Precision del clasificador con el conjunto de prueba es = 0.5904590459045904
Recall del clasificador con el conjunto de prueba es = 0.5904590459045904
F1 del clasificador con el conjunto de prueba es = 0.5904590459045904
```

Validación cruzada con ambas bases de datos

In [9]:
```
#Concatenando vectores de etiquetas

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

In [10]:

```
#Cross Validation y exactitud SVM
clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador SVM")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud NB
clf = GaussianNB()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador Naive Bayes")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud NearestCentroid
clf = NearestCentroid()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador Nearest Centroid")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#Cross Validation y exactitud DecisionTrees
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, features, ttest, cv=10)
print ("Clasificador DecisionTrees")
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='precision_micr
print (scores)
```

```
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='recall_micro')
print (scores)
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_val_score(clf, features, ttest, cv=10, scoring='f1_micro')
print (scores)
print("F1_Score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Clasificador SVM
[0.70093458 0.71495327 0.76168224 0.76168224 0.6682243  0.81690141
 0.79342723 0.75586854 0.78301887 0.73584906]
Accuracy: 0.75 (+/- 0.09)
[0.70093458 0.71495327 0.76168224 0.76168224 0.6682243  0.81690141
 0.79342723 0.75586854 0.78301887 0.73584906]
Precision: 0.75 (+/- 0.09)
[0.70093458 0.71495327 0.76168224 0.76168224 0.6682243  0.81690141
 0.79342723 0.75586854 0.78301887 0.73584906]
Recall: 0.75 (+/- 0.09)
[0.70093458 0.71495327 0.76168224 0.76168224 0.6682243  0.81690141
 0.79342723 0.75586854 0.78301887 0.73584906]
F1_Score: 0.75 (+/- 0.09)
Clasificador Naive Bayes
[0.63084112 0.61214953 0.62616822 0.57009346 0.56542056 0.73239437
 0.68544601 0.68075117 0.70754717 0.70283019]
Accuracy: 0.65 (+/- 0.11)
[0.63084112 0.61214953 0.62616822 0.57009346 0.56542056 0.73239437
 0.68544601 0.68075117 0.70754717 0.70283019]
Precision: 0.65 (+/- 0.11)
[0.63084112 0.61214953 0.62616822 0.57009346 0.56542056 0.73239437
 0.68544601 0.68075117 0.70754717 0.70283019]
Recall: 0.65 (+/- 0.11)
[0.63084112 0.61214953 0.62616822 0.57009346 0.56542056 0.73239437
 0.68544601 0.68075117 0.70754717 0.70283019]
F1_Score: 0.65 (+/- 0.11)
Clasificador Nearest Centroid
[0.68224299 0.71495327 0.74766355 0.74299065 0.64485981 0.77934272
 0.73239437 0.74178404 0.77830189 0.70754717]
Accuracy: 0.73 (+/- 0.08)
[0.68224299 0.71495327 0.74766355 0.74299065 0.64485981 0.77934272
 0.73239437 0.74178404 0.77830189 0.70754717]
Precision: 0.73 (+/- 0.08)
[0.68224299 0.71495327 0.74766355 0.74299065 0.64485981 0.77934272
 0.73239437 0.74178404 0.77830189 0.70754717]
Recall: 0.73 (+/- 0.08)
[0.68224299 0.71495327 0.74766355 0.74299065 0.64485981 0.77934272
 0.73239437 0.74178404 0.77830189 0.70754717]
F1_Score: 0.73 (+/- 0.08)
Clasificador DecisionTrees
[0.6635514  0.62149533 0.62149533 0.64485981 0.59813084 0.66666667
 0.76995305 0.68075117 0.63207547 0.6745283 ]
Accuracy: 0.66 (+/- 0.09)
[0.69158879 0.63084112 0.63551402 0.64018692 0.61682243 0.66666667
 0.77934272 0.67605634 0.65566038 0.67924528]
Precision: 0.67 (+/- 0.09)
[0.6588785  0.61682243 0.62149533 0.62616822 0.61214953 0.67605634
 0.75586854 0.64788732 0.66037736 0.69811321]
Recall: 0.66 (+/- 0.08)
[0.69158879 0.60747664 0.64485981 0.60280374 0.58878505 0.69014085
```

```
 0.76056338 0.69953052 0.68396226 0.67924528]
F1_Score: 0.66 (+/- 0.10)
```

ANEXO III. Clasificación de polaridad Emocional de Mensajes de Twitter BD #CobardeMatoncito para 3 clases

Procedimiento de clasificación

1. Preparación de la plataforma de procesamiento.
2. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base dedatos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).
3. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación yde palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.
4. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en unvector numérico para sus procesamiento.
5. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con losmensajes vectorizados y su vector de clases.
6. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica enla predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.
7. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadascon el clasificador.

1. Preparación de la plataforma de procesamiento

In [1]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from numpy import array
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

2. Integración del conjunto de entrenamiento para 3 clases: Positivo, Negativo y Neutral

In [6]:

```
#Abriendo y asignando archivos de bases de datos a arreglos (BD TASS y BD Pro

twits_train=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').Polaridad

#Concatenando Bases de mensajes y vectores de etiquetas de clases.
datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

3. Procesamiento de Texto de Bases de datos

In [22]:

```
# Abriendo archivo de BD Nueva a etiquetar:

#######----Indique archivo de BD Nueva a etiquetar----#######
twits_test_new=pd.read_csv('cobardematoncito.csv', encoding='latin-1').text
###########----------------------------------------###########

# Concatenando BD Nueva a BD de Entrenamiento:
datas = [ttrain, twits_test_new]
ttrain_new = pd.concat(datas)
#Limpiando palabras para construir vectores de tokens
#Eliminado acentos
datext = ttrain_new
#Eliminando acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

4. Vectorización TF-IDF

In [23]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:", features.shape)
#print(features.shape)
#print("")
#print("Como ejemplo se muestra el vector del primer mensaje:")
#print(features[0])
#print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
6952, 'ensenadagob': 11698, 'protecciã³ncivil': 25978, 'controlar': 8034, 'si
niestro': 29457, 'vivienda': 32905, 'afectada': 1965, 'f5ejsdcqfo': 12809, 'a
ysagonzalez': 4302, 'sostuvo': 29830, 'reuniã³n': 27865, 'sre_mx': 29939, 'ev
aluar': 12465, 'migratoria': 21176, 'mayavisiã³ntv': 20670, 'todoquev': 3110
8, 'khieoraxyx': 18139, 'derechoderã': 9458, 'plica': 25116, 'ctorvaldez': 85
04, 'eduvigesjosã': 10924, 'nezsandov': 22413, 'solicitado': 29672, 'respect
o': 27691, 'usan': 32099, 'semar': 29053, 'divertirs': 10343, 'destruir': 982
8, 'estromatolito': 12357, 's8b8ocowxo': 28553, 'cadet': 5569, 'incursionan':
16169, 'mica': 21118, 'lw4v24q2pu': 19748, 'lagranfuerzademexico': 18438, 'am
etralladora': 2758, 'cuentan': 8595, 'queretarolohacemostodo': 26544, 'puebla
informa': 26140, 'monterrey': 21586, 'cdmxinsegura': 6292, 'q1bfzuisww': 2632
1, 'enbrev': 11467, 'electo': 11131, 'ssp_puebla': 29976, 'kvwwhyyxag': 1831
8, 'incluido': 16108, 'salina': 28688, 'milla': 21237, 'utilizada': 32149, 'p
anther': 23943, 'rxs6y9emcx': 28506, 'rescata': 27640, 'tripulant': 31513, 'u
ygn0wqb1w': 32212, 'entrega': 11777, 'delphinu': 9312, 'museo': 21903, 'balle
na': 4487, 'ciencia': 6909, 'donativo': 10500, '111': 114, '119': 120, 'vaqui
ta': 32354, 'ahcancunpmeim': 2156, 'frank_lopez_': 13578, 'nauticosqroo': 221
96, 'pepecandia': 24548, 'cptqroo': 8294, 'semaqroo': 29051, 'xstiiojlmc': 33
651, 'peligro': 24435, 'poblacion': 25166, 'atã': 4073, 'presiã': 25688, 'pes
```

5. Entrenamiento del Clasificador SVM

In [24]:

```
train=features[0:(len(ttrain))]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,ttest)
pred=clsfcdr.predict(train)
print ("Exactitud de entrenamiento     = %s" % accuracy_score(ttest, pred))
print ("Precision de entrenamiento     = %s" % precision_score(ttest, pred, a
print ("Exhaustividad de entrenamiento = %s" % recall_score(ttest, pred, aver
print ("F1 de entrenamiento            = %s" % f1_score(ttest, pred, average=
```

```
Exactitud de entrenamiento     = 0.9401459854014599
Precision de entrenamiento     = 0.9401459854014599
Exhaustividad de entrenamiento = 0.9401459854014599
F1 de entrenamiento            = 0.9401459854014599
```

6. Estimación de clases de la BD Nueva por lotes

In [25]:

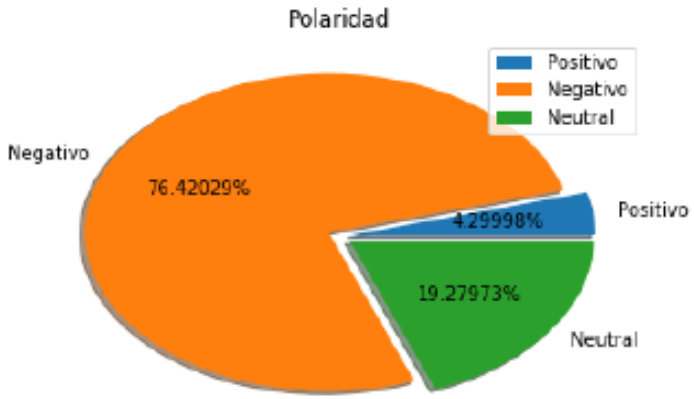
```
ti = time()

a=len(twits_test_new)/len(ttrain)

if a>3: a = int(a/3)
print ("Número de lotes a clasificar", a )

y=len(ttrain)
pred=[]
for x in range(1,a+1): #a+1
    z=1+int(len(twits_test_new)/a*x)
    print (y,z)
    test=features[y:z]
    predtest=clsfcdr.predict(test)
    pred.extend(predtest)
    print (x , "/" , a)
    y=z+1
tf=time()
tt=(tf-ti)/60
print ("Procesado en", tt, "minutos.")
```

```
Número de lotes a clasificar 6
3425 10902
1 / 6
10903 21803
2 / 6
21804 32704
3 / 6
32705 43605
4 / 6
43606 54506
5 / 6
54507 65407
6 / 6
Procesado en 182.2786380092303 minutos.
```

6. Visualización.

In [26]:
```
1 tot=len(pred)
2 sents = [pred.count(1)/tot*100, pred.count(-1)/tot*100, pred.count(0)/tot*100
3 impr= ["Positivo", "Negativo", "Neutral"]
4 expl = (0.05,0.05,0.05)
5 plt.pie(sents, explode=expl, labels=impr, autopct='%1.5f%%', shadow=True)
6 plt.title("Polaridad")
7 plt.legend()
```

Out[26]: <matplotlib.legend.Legend at 0xb353373cf8>



ANEXO IV. Clasificación de polaridad Emocional de Mensajes de Twitter BD #CobardeMatoncito para 2 clases

Procedimiento de clasificación

1. Preparación de la plataforma de procesamiento.
2. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base dedatos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).
3. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación yde palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.
4. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en unvector numérico para sus procesamiento.
5. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con losmensajes vectorizados y su vector de clases.
6. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica enla predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.
7. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadascon el clasificador.

1. Preparación de la plataforma de procesamiento

In [1]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from numpy import array
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

2. Integración del conjunto de entrenamiento para 3 clases: Positivo, Negativo y Neutral

In [2]:

```
#Abriendo y asignando archivos de bases de datos a arreglos (BD TASS y BD Pro

twits_train=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').Polaridad

#Concatenando Bases de mensajes y vectores de etiquetas de clases.
datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

3. Procesamiento de Texto de Bases de datos

In [3]:

```
# Abriendo archivo de BD Nueva a etiquetar:

#######----Indique archivo de BD Nueva a etiquetar----#######
twits_test_new=pd.read_csv('cobardematoncito.csv', encoding='latin-1').text
###########-------------------------------------###########

# Concatenando BD Nueva a BD de Entrenamiento:
datas = [ttrain, twits_test_new]
ttrain_new = pd.concat(datas)
#Limpiando palabras para construir vectores de tokens
#Eliminado acentos
datext = ttrain_new
#Eliminando acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

4. Vectorización TF-IDF

In [4]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:", features.shape)
#print(features.shape)
#print("")
#print("Como ejemplo se muestra el vector del primer mensaje:")
#print(features[0])
#print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
El tamaño del corpus vectorizado es: (67539, 30483)
El diccionario generado y su frecuencia se muestra a continuación:

{'myendlesshazza': 19527, 'puto': 23363, 'mal': 17765, 'escribo': 10706, 'sig
o': 26106, 'surrando': 26899, 'help': 13451, 'quedado': 23536, 'raro': 23891,
'cometelo': 6531, 'ahi': 1861, 'jajajaja': 15173, 'estherct209': 10921, 'much
a': 19384, 'gent': 12547, 'seguro': 25800, 'puedo': 23263, 'melena': 18506,
'muero': 19412, 'vale': 28651, 'visto': 29187, 'tia': 27456, 'bebiendos': 410
7, 'regla': 24340, 'hs': 13753, 'dado': 7798, 'much': 19383, 'grima': 12925,
'terminado': 27350, 'rio2016': 24959, 'lamentablement': 16431, 'arriendo': 32
27, 'ganancia': 12370, 'pueblo': 23257, 'brasileño': 4562, 'penuria': 21819,
'espera': 10810, 'suert': 26780, 'solidaridad': 26390, 'toni_end': 27697, 'se
ria': 25949, 'mejor': 18493, 'dejasen': 8161, 'emitir': 10087, 'basura': 403
3, 'evolucionar': 11108, 'bien': 4266, 'jonoro96': 15617, 'mandaria': 17866,
'comprart': 6660, 'burro': 4730, 'creo': 7370, 'tienda': 27479, 'abierta': 11
89, 'ahora': 1871, 'mg': 18748, 'pongo': 22527, 'adjetivo': 1562, 'super': 26
838, 'repelent': 24475, 'nombr': 20084, 'hywzz': 13885, 'voz': 29353, 'mari
a': 18082, 'final': 11787, 'mata': 18276, 'jajajsjajajaj': 15220, 'quilla': 2
3642, 'mendescod': 18535, 'heroinseb': 13487, 'enserio': 10386, 'habei': 1316
```

5. Entrenamiento del Clasificador SVM

In [5]:

```
train=features[0:(len(ttrain))]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,ttest)
pred=clsfcdr.predict(train)
print ("Exactitud de entrenamiento     = %s" % accuracy_score(ttest, pred))
print ("Precision de entrenamiento     = %s" % precision_score(ttest, pred, a
print ("Exhaustividad de entrenamiento = %s" % recall_score(ttest, pred, aver
print ("F1 de entrenamiento            = %s" % f1_score(ttest, pred, average=
```

```
Exactitud de entrenamiento     = 0.980309423347398
Precision de entrenamiento     = 0.980309423347398
Exhaustividad de entrenamiento = 0.980309423347398
F1 de entrenamiento            = 0.980309423347398
```

6. Estimación de clases de la BD Nueva por lotes

In [6]:

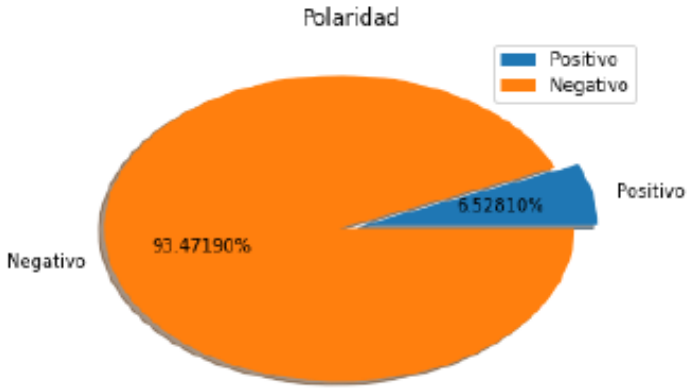
```
ti = time()

a=len(twits_test_new)/len(ttrain)

if a>3: a = int(a/3)
print ("Número de lotes a clasificar", a )

y=len(ttrain)
pred=[]
for x in range(1,a+1): #a+1
    z=1+int(len(twits_test_new)/a*x)
    print (y,z)
    test=features[y:z]
    predtest=clsfcdr.predict(test)
    pred.extend(predtest)
    print (x , "/" , a)
    y=z+1
tf=time()
tt=(tf-ti)/60
print ("Procesado en", tt, "minutos.")
```

```
Número de lotes a clasificar 10
2133 6541
1 / 10
6542 13082
2 / 10
13083 19622
3 / 10
19623 26163
4 / 10
26164 32704
5 / 10
32705 39244
6 / 10
39245 45785
7 / 10
45786 52325
8 / 10
52326 58866
9 / 10
58867 65407
10 / 10
Procesado en 83.54931745529174 minutos.
```

6. Visualización.

In [7]:
```
1 tot=len(pred)
2 sents = [pred.count(1)/tot*100, pred.count(-1)/tot*100]
3 impr= ["Positivo", "Negativo"]
4 expl = (0.05,0.05)
5 plt.pie(sents, explode=expl, labels=impr, autopct='%1.5f%%', shadow=True)
6 plt.title("Polaridad")
7 plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0x366048a400>



ANEXO V. Clasificación de polaridad Emocional de Mensajes de Twitter BD #PrensaProstituida para 3 clases

Procedimiento de clasificación

1. Preparación de la plataforma de procesamiento.
2. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base de datos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).
3. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación y de palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.
4. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en unvector numérico para sus procesamiento.
5. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con losmensajes vectorizados y su vector de clases.
6. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica en la predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.
7. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadascon el clasificador.

1. Preparación de la plataforma de procesamiento

In [2]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from numpy import array
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

2. Integración del conjunto de entrenamiento para 3 clases: Positivo, Negativo y Neutral

In [3]:

```
#Abriendo y asignando archivos de bases de datos a arreglos (BD TASS y BD Pro

twits_train=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_3.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol.csv', encoding='latin-1').Polaridad

#Concatenando Bases de mensajes y vectores de etiquetas de clases.
datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

3. Procesamiento de Texto de Bases de datos

In [4]:

```
# Abriendo archivo de BD Nueva a etiquetar:

#######----Indique archivo de BD Nueva a etiquetar----#######
twits_test_new=pd.read_csv('PrensaProstituida.csv', encoding='latin-1').text
############----------------------------------------############

# Concatenando BD Nueva a BD de Entrenamiento:
datas = [ttrain, twits_test_new]
ttrain_new = pd.concat(datas)
#Limpiando palabras para construir vectores de tokens
#Eliminado acentos
datext = ttrain_new
#Eliminando acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

4. Vectorización TF-IDF

In [5]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:", features.shape)
#print(features.shape)
#print("")
#print("Como ejemplo se muestra el vector del primer mensaje:")
#print(features[0])
#print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
xcelentelun': 8288, 'documentada': 6898, 'editada': 7152, 'xicoâ': 22349, 'h2
p3vmaew9': 9818, 'leerestã': 12295, 'demoda': 6143, '8zzzdeywtf': 636, 'vuitd
idgip': 22085, 'clausuran': 4580, 'bota': 3196, 'claudiashein': 4571, 'jefa':
11402, 'quebrã³': 17593, 'rarã': 17876, 'muri': 14352, 'esclavo': 7882, 'secp
ompeo': 19254, 'bslfw7iv1d': 3284, 'qh8jaisuk': 17546, 'referencia': 18164,
 'patricia': 15888, 'escamilla': 7867, 'hamm': 9927, 'nuã': 14986, 'actora':
 993, 'amã': 1776, 'kvoiwdqrb1': 12001, '1st': 158, 'capt': 3762, 'brenda': 3
231, 'martinez': 13418, 'amp': 1757, 'assign': 2382, 'usnscomfort': 21461, 'c
onduct': 5001, 'dental': 6180, 'exam': 8283, 'ecuador': 7119, 'comfort': 479
2, 'partner': 15819, 'centralamerica': 4096, 'southamerica': 19903, 'caribbea
n': 3819, 'provid': 17270, 'dure': 7045, 'month': 14140, 'deploy': 6201, 'end
uringpromis': 7598, 'apxy19vcg1': 2099, 'egresado': 7219, 'mater': 13489, 'fe
stejarl': 8726, 'rendirl': 18300, 'ptcmx': 17340, 'lillytellez': 12499, 'baja
ferri': 2735, 'usembassymex': 21454, 'oiijnbioji': 15201, 'continuacià³n': 52
13, 'presentacià³n': 16995, 'abogado': 806, 'charrã': 4214, 'ctor': 5606, 'de
lgado': 6110, 'presentaron': 17002, 'lupita': 12917, 'gonzã': 9555, 'lez': 12
405, 'central': 4095, 'mulzlsljs4': 14339, 'invitamo': 11064, 'consultar': 51
59, 'gons3zcbr8': 9554, 'quzbuvsla5': 17732, 'humillada': 10302, 'inmediato':
10831, 'amarrada': 1640, 'legalment': 12300, 'defenders': 6031, 'kphfa7bppw':
```

5. Entrenamiento del Clasificador SVM

In [6]:

```
train=features[0:(len(ttrain))]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,ttest)
pred=clsfcdr.predict(train)
print ("Exactitud de entrenamiento     = %s" % accuracy_score(ttest, pred))
print ("Precision de entrenamiento     = %s" % precision_score(ttest, pred, a
print ("Exhaustividad de entrenamiento = %s" % recall_score(ttest, pred, aver
print ("F1 de entrenamiento            = %s" % f1_score(ttest, pred, average=
```

```
Exactitud de entrenamiento     = 0.9389781021897811
Precision de entrenamiento     = 0.9389781021897811
Exhaustividad de entrenamiento = 0.9389781021897811
F1 de entrenamiento            = 0.9389781021897811
```

6. Estimación de clases de la BD Nueva por lotes

In [7]:

```
ti = time()

a=len(twits_test_new)/len(ttrain)

if a>3: a = int(a/3)
print ("Número de lotes a clasificar", a )

y=len(ttrain)
pred=[]
for x in range(1,a+1): #a+1
    z=1+int(len(twits_test_new)/a*x)
    print (y,z)
    test=features[y:z]
    predtest=clsfcdr.predict(test)
    pred.extend(predtest)
    print (x , "/" , a)
    y=z+1
tf=time()
tt=(tf-ti)/60
print ("Procesado en", tt, "minutos.")
```

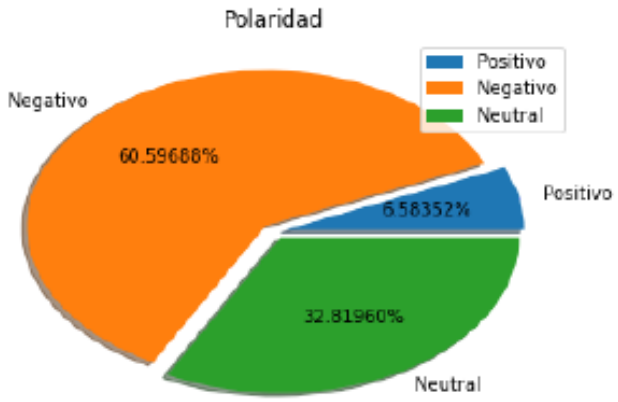
```
Número de lotes a clasificar 1
3425 14650
1 / 1
Procesado en 21.901940870285035 minutos.
```

6. Visualización.

In [8]:

```
tot=len(pred)
sents = [pred.count(1)/tot*100, pred.count(-1)/tot*100, pred.count(0)/tot*100
impr= ["Positivo", "Negativo", "Neutral"]
expl = (0.05,0.05,0.05)
plt.pie(sents, explode=expl, labels=impr, autopct='%1.5f%%', shadow=True)
plt.title("Polaridad")
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0x4665875c18>



ANEXO VI. Clasificación de polaridad Emocional de Mensajes de Twitter BD #PrensaProstituida para 2 clases

Procedimiento de clasificación

1. Preparación de la plataforma de procesamiento.
2. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base dedatos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).
3. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación yde palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.
4. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en unvector numérico para sus procesamiento.
5. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con losmensajes vectorizados y su vector de clases.
6. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica enla predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.
7. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadascon el clasificador.

1. Preparación de la plataforma de procesamiento

In [1]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from numpy import array
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

2. Integración del conjunto de entrenamiento para 2 clases: Positivo y Negativo

In [2]:

```
#Abriendo y asignando archivos de bases de datos a arreglos (BD TASS y BD Pro

twits_train=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').Polaridad

#Concatenando Bases de mensajes y vectores de etiquetas de clases.
datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

3. Procesamiento de Texto de Bases de datos

In [3]:

```
# Abriendo archivo de BD Nueva a etiquetar:

#######---Indique archivo de BD Nueva a etiquetar----#######
twits_test_new=pd.read_csv('PrensaProstituida.csv', encoding='latin-1').text
###########-----------------------------------##########

# Concatenando BD Nueva a BD de Entrenamiento:
datas = [ttrain, twits_test_new]
ttrain_new = pd.concat(datas)
#Limpiando palabras para construir vectores de tokens
#Eliminado acentos
datext = ttrain_new
#Eliminando acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

4. Vectorización TF-IDF

In [4]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:", features.shape)
#print(features.shape)
#print("")
#print("Como ejemplo se muestra el vector del primer mensaje:")
#print(features[0])
#print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
11757, 'gent': 7675, 'seguro': 15915, 'puedo': 14370, 'melena': 11213, 'muer
o': 11774, 'vale': 17701, 'visto': 18030, 'tia': 16981, 'bebiendos': 2308, 'r
egla': 15040, 'hs': 8411, 'dado': 4713, 'much': 11756, 'grima': 7918, 'termin
ado': 16911, 'rio2016': 15388, 'lamentablement': 9967, 'arriendo': 1784, 'gan
ancia': 7583, 'pueblo': 14360, 'brasileño': 2575, 'penuria': 13231, 'espera':
6560, 'suert': 16549, 'solidaridad': 16317, 'toni_end': 17147, 'seria': 1601
9, 'mejor': 11204, 'dejasen': 4933, 'emitir': 6105, 'basura': 2261, 'evolucio
nar': 6767, 'bien': 2396, 'jonoro96': 9508, 'mandaria': 10790, 'comprart': 39
47, 'burro': 2668, 'creo': 4453, 'tienda': 16994, 'abierta': 565, 'ahora': 96
3, 'mg': 11349, 'pongo': 13716, 'adjetivo': 805, 'super': 16586, 'repelent':
15114, 'nombr': 12173, 'hywzz': 8492, 'voz': 18134, 'maria': 10937, 'final':
7224, 'mata': 11068, 'jajajsjajajaj': 9293, 'quilla': 14592, 'mendescod': 112
37, 'heroinseb': 8260, 'enserio': 6287, 'habei': 8068, 'dicho': 5404, 'froo
t': 7418, 'loop': 10448, 'estan': 6619, 'malo': 10745, 'dolido': 5651, 'oye':
12787, 'luciaskaa': 10564, 'pone': 13705, 'hablar': 8090, 'josecs02': 9545,
'viejo': 17956, 'tiempo': 16993, 'avisa': 2070, 'juanalfonso251': 9593, 'cris
dazgar': 4478, 'jclilgangst': 9369, 'aguantar': 936, 'niño': 12118, 'pequeñ
o': 13243, 'peor': 13232, 'pabloserrano': 12817, 'jesusbengoechea': 9405, 'ma
drid': 10675, 'cultura': 4639, 'propia': 14214, 'vamo': 17731, 'romano': 1548
```

5. Entrenamiento del Clasificador SVM

In [5]:

```
train=features[0:(len(ttrain))]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,ttest)
pred=clsfcdr.predict(train)
print ("Exactitud de entrenamiento     = %s" % accuracy_score(ttest, pred))
print ("Precision de entrenamiento     = %s" % precision_score(ttest, pred, a
print ("Exhaustividad de entrenamiento = %s" % recall_score(ttest, pred, aver
print ("F1 de entrenamiento            = %s" % f1_score(ttest, pred, average=
```

```
Exactitud de entrenamiento     = 0.980309423347398
Precision de entrenamiento     = 0.980309423347398
Exhaustividad de entrenamiento = 0.980309423347398
F1 de entrenamiento            = 0.980309423347398
```

6. Estimación de clases de la BD Nueva por lotes

In [6]:

```
ti = time()

a=len(twits_test_new)/len(ttrain)

if a>3:
    a = int(a/3)
    print ("Número de lotes a clasificar", a )

    y=len(ttrain)
    pred=[]
    for x in range(1,a+1): #a+1
        z=1+int(len(twits_test_new)/a*x)
        print (y,z)
        test=features[y:z]
        predtest=clsfcdr.predict(test)
        pred.extend(predtest)
        print (x , "/" , a)
        y=z+1
elif a<=3:
    pred=[]
    test = features[y:]
    predtest=clsfcdr.predict(test)
    pred.extend(predtest)
    print ("Número de lotes clasificados: 1")
    print ("Tamaño de lote clasificado:", len(twits_test_new))
tf = time()
tt = (tf-ti)/60
print ("Procesado en", tt, "minutos.")
```

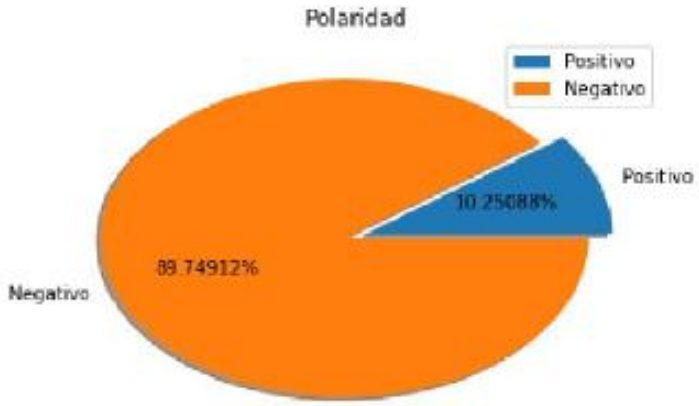
```
Número de lotes a clasificar 2
2133 7325
1 / 2
7326 14650
2 / 2
Procesado en 9.22276592652003 minutos.
```

6. Visualización.

In [7]:

```
1 tot=len(pred)
2 sents = [pred.count(1)/tot*100, pred.count(-1)/tot*100]
3 impr= ["Positivo", "Negativo"]
4 expl = (0.05,0.05)
5 plt.pie(sents, explode=expl, labels=impr, autopct='%1.5f%%', shadow=True)
6 plt.title("Polaridad")
7 plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0xf4cfecdb70>



ANEXO VII. Modelo de Prioridad basada en diccionario

Propósito. Identificar mensajes de interés a partir de palabras clave de una diccionario.

El diccionario de palabras clave se genera a partir de una BD de mensajes etiquetados como "de
interés". Para una BD de entrada se identifican y cuentan en sus mensajes palabras que pertenecen al diccionario. Aplica el siguiente procedimiento.

0. Preparación de la plataforma de procesamiento.

1. Procesamiento de texto, para generar tokens limpios a partir de la base de datos de mensajes significativos.

2. Generación de Diccionario. Utilizando la función vecorizer.vocabulary_ de la librería feature_extraction.text.CountVectorizer de ScyKit-Learn, se obtiene el diccionario de términos contenidos en la base de datos.
Posteriormente se realiza una revisión y eliminación de palabras poco signifi cativas para contar con un diccionario final. Para este caso a partir de una base de datos de 255 mensajes, proporcionados por la Institución, se obtuvo un diccionario de 555 palabras significativas.

3. Procesamiento de texto, para generar tokens limpios de una base de datos de mensajes para la cual se desea identificar los mensajes prioritarios.

4. Modelo de Frecuencia de término. Para cada uno de los vectores de tokens de la base de datos que se desea clasificar, se identifican y cuentan las palabras del diccionario que contenga el mensaje; mientras palabras significativas contenga un mensaje se le considera más prioritario.

5. Visualización y Consulta. Se muestra gráfico de frecuencia de términos para la base de mensajes. Usando funciones para gestión de arreglos es posible realizar consultas para recuperar los mensajes que contengan más palabras significativas, los cuales son considerdos de mayor prioridad.

0. Preparación de la plataforma de procesamiento

In [39]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from sklearn.feature_extraction.text import CountVectorizer
from numpy import array
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
```

1. Procesamiento de texto de la base de mensajes significativos

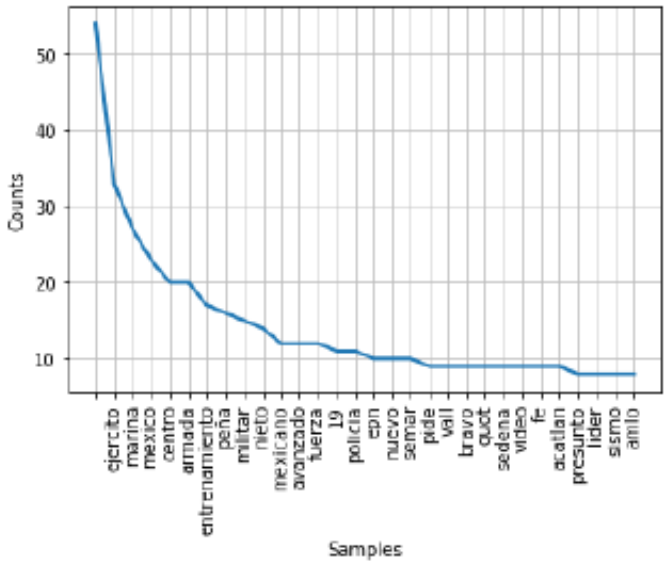
In [41]:

```
#Cargando BD de mensajes significativos

datext=pd.read_csv('semar277.csv', encoding='UTF-8').title

#Limpiando paralabras para construir duccionario
#Eliminado acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [42]:

```
# Gráfica de Zipf. Frecuencia de palabras en mensajes.
tw = []
for lin in stem:
    [tw.append(w) for w in lin] #Agrupando el total de palabras stemms en una
fdist = nltk.FreqDist(tw)
print (fdist.most_common(50))
fdist.plot(30)
```

```
[('', 54), ('ejercito', 33), ('marina', 27), ('mexico', 23), ('centro', 20),
('armada', 20), ('entrenamiento', 17), ('peña', 16), ('militar', 15), ('nieto',
14), ('mexicano', 12), ('avanzado', 12), ('fuerza', 12), ('19', 11), ('polici
a', 11), ('epn', 10), ('nuevo', 10), ('semar', 10), ('pide', 9), ('vall', 9),
('bravo', 9), ('quot', 9), ('sedena', 9), ('video', 9), ('fe', 9), ('acatlan',
9), ('presunto', 8), ('lider', 8), ('sismo', 8), ('amlo', 8), ('año', 7), ('d
o', 7), ('edomex', 6), ('frida', 6), ('caso', 6), ('detienen', 6), ('laredo',
6), ('39', 6), ('respons', 6), ('ley', 6), ('financiera', 6), ('titular', 6),
('unam', 6), ('narcomenudista', 6), ('soldado', 5), ('fortalecimiento', 5), ('d
roga', 5), ('pai', 5), ('rescatista', 5), ('inaugura', 5)]
```



2. Generación de Diccionario.

In [44]:
```
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:", features.shape)
diccionario = vectorizer.vocabulary_
diccionario
```

```
El tamaño del corpus vectorizado es: (256, 866)
```

In [45]:
```
#Se realiza limpieza manual de palabras varias, no significativas).
limpiar = ['09','15','16','16deseptiembr','170','19','1968','1985','20','200'
```

In [46]:
```
for x in limpiar:
    del diccionario[x]
```

In [47]:

```
limpiar = ['abr','abrir', 'acatlan', 'agua','alternativa', 'alto', 'amado', '
for x in limpiar:
    del diccionario[x]
diccionario
```

```
'hero': 403,
'hombr': 408,
'honor': 409,
'huachicol': 412,
'huachicolero': 413,
'humanidad': 414,
'humano': 415,
'ilicita': 417,
'ilicito': 418,
'impacto': 420,
'impotencia': 421,
'impunidad': 422,
'incendio': 426,
'incumplen': 427,
'indagatoria': 428,
'independencia': 429,
'indigena': 430,
'indispens': 431,
'industria': 432,
'inform': 433.
```

Clasificando Twits por prioridad por diccionario

3. Procesamiento de texto a base de datos a clasificar.

In [61]:

```
#Cargando archivo
twits_set=pd.read_csv('semar70120.csv', encoding='latin-1').text

#limpiando mensajes
#Eliminado acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in twits_set:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)

#Eliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [70]:

```
print ("Mensajes que se analizan:" , len(stems))
```

```
Mensajes que se analizan: 3717
```

4. Modelo de frecuencia de término basado en diccionario.

In [63]:
```
#Identifica y cuenta palabras significativas en cada mensaje.
etik =[]
i=0
for lin in stem:
    for w in lin:
        if w in diccionario: i=i+1
    etik.append(i)
    i=0
etik = np.array(etik)
```

In [64]:
```
#Imprimiento vector de frecuencia de términos significativos
print (etik)
```

```
[1 3 1 ... 0 1 1]
```

In [65]:
```
#Imprimiento vectores de tokens y frecuencias de términos significativos
i=0
for lin in stem:
    print (lin, etik[i])
    i=1+1
```

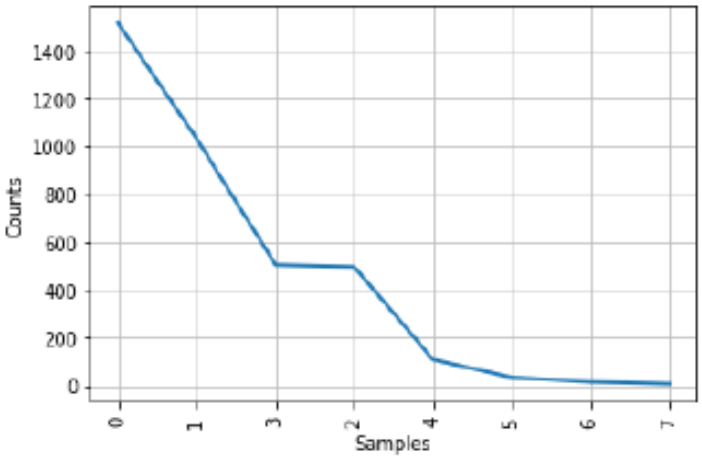
```
['rt', 'mmorena2021', 'el_universal_mx', 'filtra', 'primera', 'fotografã', 'n
arco', 'gobierno', 'felip', 'calderã', 'n', 'chapo', 'controlaba', 'diâ', ''] 
1
['rt', 'mmorena2021', 'reforma', 'alerta', 'â', '%ï', 'primera', 'fotografã',
'narco', 'gobierno', 'felip', 'calderã', 'n', 'chapo', 'â', 'protecciã³n', 'i
nformâ', ''] 1
['rt', 'mmorena2021', 'el_universal_mx', 'filtra', 'primera', 'fotografã', 'n
arco', 'gobierno', 'felip', 'calderã', 'n', 'chapo', 'controlaba', 'diâ', ''] 
1
['rt', 'semar_mx', 'embajador', 'caballero', 'mare', 'glorioso', 'forjador',
'mujer', 'hombr', 'mar', 'qbzjtlenuw'] 1
['', 'cuitlahuacgj', 'gn_mexico_', 'sp_veracruz', 'spcver', 'pcestatalv', 'hg
utierrez_m', 'sedenamx', 'semar_mx', 'ericcisnerosb', 'lupeosorno', 'c4_ver',
'tanta', 'mentira', 'dice', 'respeto', 'ah', 'hecho', 'bienestar', 'veracruza
no', 'mã', 's', 'pobreza', 'mã', 's', 'violencia', 'siempr', 'gobierno', 'pas
ado', 'ojalã', 'algãºn', 'dã', 'hablemo', 'cifra', 'congruent', 'real'] 1
['rt', 'mmorena2021', 'el_universal_mx', 'filtra', 'primera', 'fotografã', 'n
arco', 'gobierno', 'felip', 'calderã', 'n', 'chapo', 'controlaba', 'diâ', ''] 
1
```

Visualización y consulta.

In [66]:

```
# Imprimiendo cantidad de mensajes correspondientes a frecuencia de término s
fdist1 = nltk.FreqDist(etik)
print("Cantidad de mensajes para cada frecuencua de término significativo")
print (fdist1.most_common(maxp+1))
print("Gráfica de cantidad de mensajes para cada frecuencia de término signif
fdist1.plot(maxp+1)
```

```
Cantidad de mensajes para cada frecuencua de término significativo
[(0, 1515), (1, 1032), (3, 505), (2, 497), (4, 109), (5, 35), (6, 16), (7, 8)]
Gráfica de cantidad de mensajes para cada frecuencia de término significativo
```



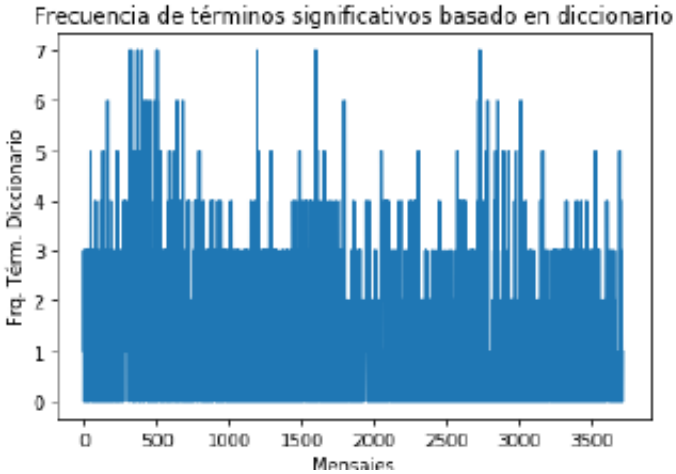
In [69]:

```
#Imprimiendo mensajes prioritarios.

maxp=np.amax(etik)
print ("Mensajes prioritarios")
print ("")

i=0
w=0
for x in etik:
    if x>(maxp-3):
        print (twits_set[i], x)
        print ("")
        w=w+1
    i=i+1
print ("")
print ("Total de mensajes prioritarios", w)
```

```
anco en Coatzacoalcos donde fallecieron 32 personas https://t.co/gZcOvcuiB2
 (https://t.co/gZcOvcuiB2) 5

SeÃ±or Presidente @lopezobrador_ la @PoliciaFedMx cuida a los huachicoleros e
n Chignahuapan puebla, el huachicol se extrae de cuautepec de Hinojosa y de a
huazotepec, puebla, es increÃble la impunidad con la que estos delincuentes o
peran, envie 6meses a la @SEMAR_mx y verÃ¡ https://t.co/d8kUM617wW (https://
t.co/d8kUM617wW) https://t.co/9YakAxhDmH (https://t.co/9YakAxhDmH) 5

Labores de inteligencia naval de @SEMAR_mx, llevaron a la captura de Daniel
 âœOâ  (a) "Comandante Meca y/o Calavera, Jefe Operativo de la cÃ©lula delict
iva de JesÃºs Adolfo BaÃ±os SalomÃ³n (a) el â€50â, Jefe de Plaza de Coatzacoa
lcos para el CJNG. https://t.co/KGSZaqCkU2 (https://t.co/KGSZaqCkU2) 6

Un dato que es parte del panorama de seguridad en #NuevoLaredo. Hace mÃ¡s de
 un aÃ±o, la @SEMAR_mx retirÃ³ a sus elementos, debido a la acusaciÃ³n de que
marinos participaron en, por lo menos, 45 desapariciones forzadas.

AcÃ¡ la nota:
https://t.co/Nj4xbBEiLe (https://t.co/Nj4xbBEiLe) 5
```

In [68]:
```
#Imprimiento gráfico de Frecuencia de términos basado en doccionario
plt.xlabel('Mensajes')
plt.ylabel('Frq. Térm. Diccionario')
plt.title('Frecuencia de términos significativos basado en diccionario')
plt.plot(etik)
```

Out[68]: [<matplotlib.lines.Line2D at 0x17a0a2b7b8>]



ANEXO VIII. Modelo híbrido para identificación de mensajes prioritarios

Procedimiento de clasificación

1. Preparación de la plataforma de procesamiento.
2. Integración del conjunto de entrenamiento, concatenando la Base de Datos TASS y la base de datos de diseño propio, así se cuenta con una mayor cantidad de mensajes etiquetados. La concatenación se realiza para dos bases de datos: una con tres clases (positivo, negativo y neutral) y otra de dos clases (positivo y negativo).
3. Procesamiento de Texto a Bases de datos. Se realiza la eliminación de signos de puntuación y de palabras vacías, así como la reducción a letras minúsculas y a raíz de las palabras (stemming), se obtiene como resultado un arreglo de vectores de palabras significativas (tokens) para la base de datos concatenada y la base de datos nueva que requiere la predicción de clases.
4. Vectorización TF-IDF. Se realiza la valoración y transformación de cada vector de tokens en un vector numérico para sus procesamiento.
5. Entrenamiento del Clasificador SVM. Se entrena la máquina de vectores de soporte con los mensajes vectorizados y su vector de clases.
6. Estimación de clases de la BD Nueva. Una vez que el clasificador está entrenado, se aplica en la predicción de clases de la base de datos nueva; debido a que las bases de datos nuevas pueden ser de varios miles de mensajes, el proceso de predicción de clases se realiza por lotes para evitar desborde de memoria.
7. Visualización. Se presentan gráficamente las proporciones de polaridad emocional estimadas con el clasificador.
8. Selección de la clase de interés (Positiva o negativa) y extracción de mensajes asociado a dicha clase.
9. Procesamiento de texto, para generar tokens limpios a partir de la base de datos de mensajes significativos.
10. Generación de Diccionario. Utilizando la función vecorizer.vocabulary_ de la librería feature_extraction.text.CountVectorizer de ScyKit-Learn, se obtiene el diccionario de términos contenidos en la base de datos. Posteriormente se realiza una revisión y eliminación de palabras poco significativas para contar con un diccionario final. Para este caso a partir de una base de datos de 255 mensajes, proporcionados por la Institución, se obtuvo un diccionario de 555 palabras significativas.
11. Procesamiento de texto, para generar tokens limpios de una base de datos de mensajes para la cual se desea identificar los mensajes prioritarios.
12. Modelo de Frecuencia de término. Para cada uno de los vectores de tokens de la clase que se desea analizar, se identifican y cuentan las palabras del diccionario que contenga el mensaje; mientras palabras significativas contenga un mensaje se le considera más prioritario.
13. Visualización y Consulta. Se muestra gráfico de frecuencia de términos para la base de mensajes. Usando funciones para gestión de arreglos es posible realizar consultas para recuperar los mensajes que contengan más palabras significativas, los cuales son considerdos de mayor prioridad.

1. Preparación de la plataforma de procesamiento

In [1]:

```
#Librerías a emplear

from sklearn import datasets
import pandas as pd
import nltk
import re
import numpy as np
from time import time
from sklearn import metrics
import matplotlib.pyplot as plt
import math
from numpy import array
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.svm  import  SVC
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import f1_score
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

2. Integración del conjunto de entrenamiento para 2 clases: Positivo y Negativo

In [2]:

```
#Abriendo y asignando archivos de bases de datos a arreglos (BD TASS y BD Pr

twits_train=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').content
twits_train_y=pd.read_csv('Intertass_Pol_2.csv', encoding='latin-1').value
twits_test=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').text
twits_test_y=pd.read_csv('BDTwitsPol_2.csv', encoding='latin-1').Polaridad

#Concatenando Bases de mensajes y vectores de etiquetas de clases.
datas = [twits_train, twits_test]
ttrain = pd.concat(datas)

datast = [twits_train_y, twits_test_y]
ttest = pd.concat(datast)
```

3. Procesamiento de Texto de Bases de datos

In [3]:

```
# Abriendo archivo de BD Nueva a etiquetar:

#######----Indique archivo de BD Nueva a etiquetar----#######
twits_test_new=pd.read_csv('semar70120.csv', encoding='latin-1').text
###########-----------------------------------###########

# Concatenando BD Nueva a BD de Entrenamiento:
datas = [ttrain, twits_test_new]
ttrain_new = pd.concat(datas)
#Limpiando palabras para construir vectores de tokens
#Eliminado acentos
datext = ttrain_new
#Eliminando acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

4. Vectorización TF-IDF

In [4]:

```
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(stems).toarray()
print("El tamaño del corpus vectorizado es:", features.shape)
#print(features.shape)
#print("")
#print("Como ejemplo se muestra el vector del primer mensaje:")
#print(features[0])
#print("")
print("El diccionario generado y su frecuencia se muestra a continuación:")
print("")
print(vectorizer.vocabulary_)
```

```
p': 4576, 'ps': 10762, 'exterminen': 5338, 'salud': 11835, 'mental': 8557, 'p
a': 9724, 'pueda': 10801, 'dejar': 3879, 'call': 2226, 'jajajaj': 7203, 'kost
o_trad': 7563, 'juanrallo': 7417, 'prohibir': 10673, 'casco': 2458, 'ir': 709
4, 'enmascarado': 4880, 'tambien': 12649, 'arbitrario': 1332, 'sectario': 119
34, 'otegi': 9679, 'inhabilitado': 6896, 'eslasentencia': 5094, 'mujer': 894
6, 'tapada': 12666, 'ojo': 9502, 'essucultura': 5148, 'tenei': 12756, 'inamo
v': 6758, 'vida': 13625, 'lleno': 7917, 'comida': 3004, 'crush': 3546, 'lleve
i': 7935, 'albueto': 872, 'ferderer2': 5528, 'tipsterapuesta': 12894, 'rey':
11533, 'mago': 8178, 'real': 11140, 'sirdeschain': 12227, 'chicacascabel': 26
70, 'reptarazul': 11400, 'ta': 12612, 'aprecio': 1286, 'raistlin': 11029, 'da
ba': 3713, 'iwanancho': 7164, 'linoon': 7867, 'tambor': 12652, 'velocidad': 1
3505, 'extrema': 5357, 'especial': 5111, 'generacion': 5927, 'pasada': 9886,
'royaldarjeel': 11685, 'geralt': 5947, 'rivia': 11598, 'meno': 8549, 'sergios
mil': 12072, 'mirat': 8726, 'gray': 6115, 'man': 8224, 'cero': 2581, 'alguie
n': 949, 'apuntando': 1304, 'ventana': 13535, 'laser': 7669, 'defin': 3858,
'pesadito': 10134, 'carlotalotalota': 2420, 'apena': 1231, 'pasado': 9887, 'i
dea': 6614, 'crear': 3487, 'colectivo': 2936, 'estat': 5168, 'division': 438
5, 'local': 7962, 'poderoso': 10337, 'lucia': 8070, 'vd': 13482, 'amor': 109
4, 'hacia': 6275, 'gran': 6095, 'tapadera': 12667, 'comer': 2990, 'nutella':
9351, 'casa': 2453, 'entiendo': 4931, 'cree': 3497, 'realidad': 11145, 'sud
```

5. Entrenamiento del Clasificador SVM

In [5]:

```
train=features[0:(len(ttrain))]
clsfcdr=SVC(kernel='linear', C=1)
clsfcdr.fit(train,ttest)
pred=clsfcdr.predict(train)
print ("Exactitud de entrenamiento     = %s" % accuracy_score(ttest, pred))
print ("Precision de entrenamiento     = %s" % precision_score(ttest, pred, a
print ("Exhaustividad de entrenamiento = %s" % recall_score(ttest, pred, aver
print ("F1 de entrenamiento            = %s" % f1_score(ttest, pred, average=
```

```
Exactitud de entrenamiento     = 0.9784341303328645
Precision de entrenamiento     = 0.9784341303328645
Exhaustividad de entrenamiento = 0.9784341303328645
F1 de entrenamiento            = 0.9784341303328645
```

6. Estimación de clases de la BD Nueva por lotes

In [28]:

```
ti = time()

a=len(twits_test_new)/len(ttrain)

if a>3:
    a = int(a/3)
    print ("Número de lotes a clasificar", a )

    y=len(ttrain)
    pred=[]
    for x in range(1,a+1): #a+1
        z=1+int(len(twits_test_new)/a*x)
        print (y,z)
        test=features[y:z]
        predtest=clsfcdr.predict(test)
        pred.extend(predtest)
        print (x , "/" , a)
        y=z+1
elif a<=3:
    pred=[]
    test = features[y:]
    predtest=clsfcdr.predict(test)
    pred.extend(predtest)
    print ("Número de lotes clasificados: 1")
    print ("Tamaño de lote clasificado:", len(twits_test_new))
tf = time()
tt = (tf-ti)/60
print ("Procesado en", tt, "minutos.")
```

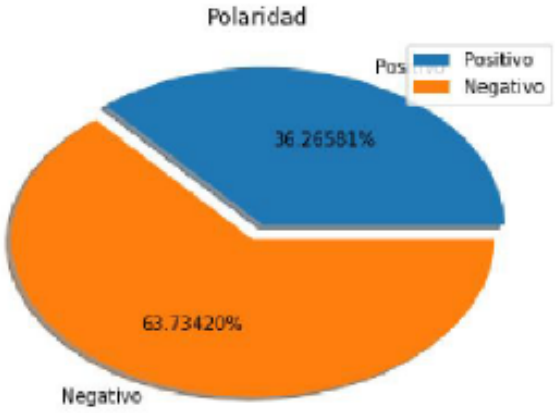
```
Número de lotes clasificados: 1
Tamaño de lote clasificado: 3717
Procesado en 2.075610319773356 minutos.
```

7. Visualización.

In [29]:

```
tot=len(pred)
sents = [pred.count(1)/tot*100, pred.count(-1)/tot*100]
impr= ["Positivo", "Negativo"]
expl = (0.05,0.05)
plt.pie(sents, explode=expl, labels=impr, autopct='%1.5f%%', shadow=True)
plt.title("Polaridad")
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x587699fc88>



8. Selección de la clase de interés

In [74]:

```
#Asignando mensajes negativas a la lista "neg[]"
neg=[]
i=0
for cls in pred:
    if cls==-1:
        neg.append(twits_test_new[i])
    i=i+1
print ("El número de mensajes negativos es:", len(neg))
```

```
El número de mensajes negativos es: 2369
```

9. Procesamiento de texto.

In [75]:

```
#Cargando BD de mensajes significativos

datext=pd.read_csv('semar277.csv', encoding='UTF-8').title

#Limpiando paralabras para construir duccionario
#Eliminado acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in datext:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)
#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)
#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear
#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)

```

10. Generación de Diccionario.

In [77]:

```
features = vectorizer.fit_transform(stems).todense()
print("El tamaño del corpus vectorizado es:", features.shape)
diccionario = vectorizer.vocabulary_

limpiar = ['09','15','16','16deseptiembr','170','19','1968','1985','20','200'
for x in limpiar:
    del diccionario[x]
```

```
El tamaño del corpus vectorizado es: (256, 866)
```

11. Procesamiento de Texto.

In [78]:

```
#limpiando mensajes
#Eliminado acentos
a,b = 'áéíóúü','aeiouu'
trans = str.maketrans(a,b)
dat=[]
for l in neg:
    l = l.lower()
    l = l.translate(trans)
    dat.append(l)

#Generando Tokens
tokens=[]
for lin in dat:
    toks = nltk.word_tokenize(lin)
    toks = re.compile(r'\W+', re.UNICODE).split(lin)
    tokens.append(toks)

#Eiliminando stopwords
fw =[]
for lin in tokens:
    filtered_words = [word for word in lin if word not in stopwords.words('sp
    fw.append(filtered_words)
    filtered_words.clear

#Realizando recorte a palabras base
ps = PorterStemmer()
stem=[]
for lin in fw:
    stemmers = [ps.stem(word) for word in lin]
    stem.append(stemmers)
    stemmers.clear
#Convirtiendo listas a vectores
stems = []
for x in stem:
    stems.append(" ".join(x))
stems = np.array(stems)
```

In [79]:

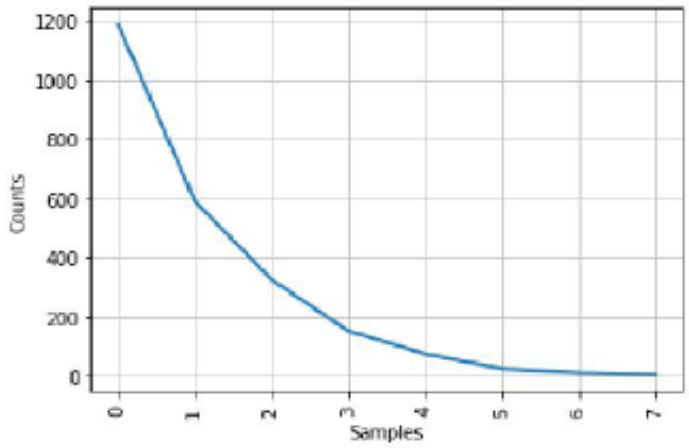
```
#Identifica y cuenta palabras significativas en cada mensaje.
etik =[]
i=0
for lin in stem:
    for w in lin:
        if w in diccionario: i=i+1
    etik.append(i)
    i=0
etik = np.array(etik)
```

12. Modelo de frecuencia de diccionario.

In [86]:

```
# Imprimiendo cantidad de mensajes correspondientes a frecuencia de término s
maxp=np.amax(etik)
fdist1 = nltk.FreqDist(etik)
print ("")
print("     Cantidad de mensajes para cada frecuencia de término significativ
print ("")
print (fdist1.most_common(maxp+1))
print ("")
print("Gráfica de cantidad de mensajes para cada frecuencia de término signif
fdist1.plot(maxp+1)
```

```
     Cantidad de mensajes para cada frecuencia de término significativo

[(0, 1186), (1, 591), (2, 324), (3, 152), (4, 75), (5, 25), (6, 11), (7, 5)]

Gráfica de cantidad de mensajes para cada frecuencia de término significativo
```



13. Visualización.

In [83]:

```
1
2  print ("Mensajes prioritarios")
3  print ("")
4
5  i=0
6  w=0
7  for x in etik:
8      if x>(maxp-2):
9          print (neg[i], x)
10         print ("")
11         w=w+1
12     i=i+1
13 print ("")
14 print ("Total de mensajes prioritarios", w)
```

```
Mensajes prioritarios

RT @VigilantesV: Detienen elementos de la @SEMAR_mx a presunto jefe del CJNG, l
igado al ataque al bar Caballo Blanco ubicado en Coatzacoalcâ□¦ 6

@lopezobrador_ @Pemex @SEDENAmx @SEMAR_mx @LuisaAlcalde @STPS_mx @SSPCMexico @S
EGOB_mx @M_OlgaSCordero @SNietoCastillo @Hacienda_Mexico #ConferenciaPresidente
| @SNietoCastillo, titular de la Unidad de Inteligencia Financiera de la secret
aria de
@Hacienda_Mexico, dijo que se han presentado dos denuncias en contra de Romero
Deschamps y que se solicitÃ³ el aseguramiento de las cuentas bancarias y bienes
inmuebles. https://t.co/J1dcnTClYe (https://t.co/J1dcnTClYe) 7

@lopezobrador_ @Pemex @SEDENAmx @SEMAR_mx @LuisaAlcalde @STPS_mx @SSPCMexico @S
EGOB_mx @M_OlgaSCordero #ConferenciaPresidente | Sobre los Legionarios de Crist
o, @SNietoCastillo, titular de la Unidad de Inteligencia Financiera de la secre
taria de @Hacienda_Mexico, dijo que, hasta el momento,  "no hay ninguna investi
gaciÃ³n que compruebe ningÃºn caso". https://t.co/otXiRhVvom (https://t.co/otXi
RhVvom) 7

@lopezobrador_ @Pemex @SEDENAmx @SEMAR_mx #ConferenciaPresidente | El president
e @lopezobrador_ seÃ±alÃ³ que se tiene como hipÃ³tesis que el  incremento del r
obo de gas a @Pemex es causado por sus competidores con la finalidad de debilit
ar  su capacidad operativa. https://t.co/MMTJtPgATG (https://t.co/MMTJtPgATG) 6

Hubo intervenciÃ³n del personal de la @SEDENAmx y de la @SEMAR_mx en las instal
aciones de @Pemex para crear una red de inteligencia y asÃ detener el robo de c
ombustible. RJ https://t.co/5LNUIEKxZ2 (https://t.co/5LNUIEKxZ2) 6

RT @VigilantesV: Detienen elementos de la @SEMAR_mx a presunto jefe del CJNG, l
igado al ataque al bar Caballo Blanco ubicado en Coatzacoalcâ□¦ 6

RT @VigilantesV: Detienen elementos de la @SEMAR_mx a presunto jefe del CJNG, l
igado al ataque al bar Caballo Blanco ubicado en Coatzacoalcâ□¦ 6

RT @VigilantesV: Detienen elementos de la @SEMAR_mx a presunto jefe del CJNG, l
igado al ataque al bar Caballo Blanco ubicado en Coatzacoalcâ□¦ 6
```

Junto con Ã©l fue detenida otra persona, ambos portaban bolsas con dosis de dro
ga

#Veracruz #verfollow https://t.co/cRXBAWFlDs (https://t.co/cRXBAWFlDs) 7

#GUAYMAS Hombres armados dispararon esta noche contra un elemento de la policÃa
municipal.
El agente recibiÃ³ al llegar a su casa 8 impactos de bala.
Se trata de Eduardo Manjarrez, quien hace un mes fue golpeado por elementos de
 @SEMAR_mx https://t.co/snvpam8AeC (https://t.co/snvpam8AeC) 7

@el_pais AquÃ el jefe de Genaro Garcia Luna el NARCO A CARGO DE LA SEGURIDAD DE
MÃ□XICO POR 6 AÃ□OS

Su jefe el Narco FELIPE CALDERÃ□N COBRÃ□ MILLONES DE DÃ□LARES dando protecciÃ³
n y atacando a los enemigos del Cartel de Sinaloa

MANDO A MORIR CIVILES SOLDADOS Y POLICÃ□AS @SEDENAmx @SEMAR_mx https://t.co/vCk
ff3PwRr (https://t.co/vCkff3PwRr) 7

Labores de inteligencia naval de @SEMAR_mx, llevaron a la captura de Daniel
 â□□Oâ□□ (a) "Comandante Meca y/o Calavera, Jefe Operativo de la cÃ©lula deli
ctiva de JesÃºs Adolfo BaÃ±os SalomÃ³n (a) el â□□50â□□, Jefe de Plaza de Coatz
acoalcos para el CJNG. https://t.co/KGSZaqCkU2 (https://t.co/KGSZaqCkU2) 6

@GobiernoMX Haz un plan cuya MISIÃ□N sea ACABAR CON LA IMPUNIDAD y contenga 2 v
ertientes:
1a. Uso de fuerza legÃtima del Estado por @SSPCMexico @SEDENAmx @SEMAR_mx y res
to de policÃas VS delincuentes.
2a. Con @CONAGO_oficial @SCJN @CJF_Mx construir efectivo sistema de procuraciÃ³
n de justicia. 6

@MMorena2021 @Rogelio13559696 @El_Universal_Mx @SEDENAmx @SEMAR_mx Todo con el
 argumento de proporcionar seguridad a los mexicanos, obtener y ejercer presupu
esto, y como dice usar la para,..., para ..., elementos que se suman a las cifr
as
,... 200 mil muerto, 35 desaparecidos, comunidades desplazadas. Es aberrante. 6

@JoelZevach @JLozanoA @FelipeCalderon @SSPCMexico @SEDENAmx @SEMAR_mx @PoliciaF
edMx @GN_MEXICO_ Solo le recuerdo que el combate al crimen organizado es de Ãnd
ole federal, le corresponde al viejo decrepito combatir ese problema y lo que h
izo fue retirar a la Marina de las calles 6

@LPueblo2 @SEMAR_mx @SEDENAmx lo que hace falta. es no tomar prisioneros Vivos.
Eso de zllenar penales de terroristas y darle de tragar gratis, pa mi como ciud
adano y que les@doy@de@tragar con@mis@impuestos, Pos no me sale. Hay que Mandar
los A descansar de x vida. y Desechos Humanos â□□ð□□» (ese de NL tbn) 6

Total de mensajes prioritarios 16

In [87]:
```
1 plt.xlabel('Mensajes')
2 plt.ylabel('Frq. Térm. Diccionario')
3 plt.title('Frecuencia de términos significativos basado en diccionario')
4 plt.plot(etik)
```

Out[87]: [<matplotlib.lines.Line2D at 0x583f04df98>]