Categorización de texto usando técnicas de Machine Learning aplicado a la clasificación de reclamos en los procesos de la Universidad Tecnológica de Bolívar

Mayo de 2013

Jorge Andrés Florián Noriega

Categorización de texto usando técnicas de Machine Learning aplicado a la clasificación de reclamos en los procesos de la Universidad Tecnológica de Bolívar.

Jorge Andrés Florián Noriega

Universidad Tecnológica de Bolívar

Programa de ingeniería de sistemas

Cartagena

Mayo de 2013

Categorización de texto usando técnicas de Machine Learning aplicado a la clasificación de reclamos en los procesos de la Universidad Tecnológica de Bolívar.

Jorge Andrés Florián Noriega

DIRECTOR

William Caicedo Torres

M.Sc. en Ingeniería

Universidad Tecnológica de Bolívar

Programa de ingeniería de sistemas

Cartagena

Mayo de 2013

Cartagena de Indias, Mayo 03 de 2013

Magister
Gonzalo Garzón
Director Programa de Ingeniería de Sistemas
Universidad Tecnológica de Bolívar

Asunto: Carta de presentación de Trabajo de Grado

Cordial saludo,

Me dirijo a usted, para hacerle entrega del Trabajo de Grado de Pregrado titulado: **"Categorización de texto usando técnicas de Machine Learning aplicado a la clasificación de reclamos en los procesos de la Universidad Tecnológica de Bolívar"**, para lo cual presento a usted original y copia del proyecto, para su evaluación por parte del comité revisor.

Por su parte, el director del trabajo, William Caicedo Torres, considera que el trabajo reúne los requisitos mínimos y la estructura académica básica para ser presentado a evaluación.

Agradezco de antemano la atención a la solicitud.

Atentamente,

_____	_____
Jorge Andrés Florián Noriega	William Caicedo Torres, MSc.
Autor	Director

Nota de aceptación:

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma jurado

_____

Firma jurado

Cartagena de Indias, Mayo 03 de 2013

DEDICATORIA

A mi padre, que se sacrifica tanto para darnos todo lo necesario.

A mi hermano por brindarme compañía. Y a mis profesores,

En especial a mi primera maestra que me enseño

A siempre seguir adelante.

Gracias mamá!

Jorge Andrés Florián Noriega

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

William Caicedo Torres, Ingeniero de sistemas, por su incondicional apoyo en el desarrollo del presente trabajo de grado, por el tiempo dedicado y bien intencionadas recomendaciones.

Al cuerpo de docentes del programa de Ingeniería de Sistemas de la Universidad Tecnológica de Bolívar, por fortalecer nuestras bases como ingenieros y brindarnos formación profesional.

CONTENIDO

1. OBJETIVOS 13

1.1 OBJETIVO GENERAL 13

1.2 OBJETIVOS ESPECÍFICOS 13

2. PROBLEMA DE INVESTIGACIÓN 14

2.1 DESCRIPCIÓN DEL PROBLEMA 14

2.2 FORMULACIÓN DEL PROBLEMA. 15

3. JUSTIFICACIÓN 16

4. ASPECTOS METODOLOGICOS. 17

4.1 TIPO DE INVESTIGACIÓN 17

4.2 METODOLOGIA DE TRABAJO 17

5. MARCO REFERENCIAL 19

5.1 MARCO CONCEPTUAL 19

5.2 MARCO TEORICO 20

5.2.1 Machine Learning. 20

5.2.2 Procesamiento del lenguaje natural. 22

5.2.3 Clasificación de texto. 24

5.2.4 Sistema de gestión de la calidad ISO 9001. 24

6. ESTADO DEL ARTE 27

7. DEFINICIÓN FORMAL DEL PROBLEMA 31

8. CONSTRUCCIÓN DE CONJUNTOS DE DATOS 32

8.1 CATEGORIZACIÓN DE LOS RECLAMOS 32

8.2 DIVISIÓN DEL CONJUNTO DE DATOS. 34
9. PROCESAMIENTO DE TEXTO. 37
10. VECTORIZACIÓN DE LOS RECLAMOS. 38
10.1 BAG OF WORDS. 38
10.2 TF-IDF. 40
11. FEATURE SELECTION. 42
11.1 SEPARACION BI-NORMAL. 44
12. METODOS DE EVALUACIÓN DE LOS CLASIFICADORES. 46
12.1 PRECISION – RECALL. 47
12.2 MEDIDA F 48
13. CONSTRUCCIÓN DE LOS MODELOS DE MACHINE LEARNING. 50
13.1 DESCRIPCIÓN DE LOS CLASIFICADORES DE MACHINE LEARNING. . 50
13.1.1 Naive Bayes. 50
13.1.2 Máquinas de Soporte Vectorial. 55
13.2 IMPLEMENTACION DE LOS MODELOS DE MACHINE LEARNING. 61
14. ENTRENAMIENTO Y EVALUACIÓN. 64
14.1 ENTRENAMIENTO 64
14.2 EVALUACIÓN 69
14.3 ANÁLISIS DE RESULTADOS. 71
15. CONCLUSIONES 74
REFERENCIAS 76
ANEXOS 83

LISTAS DE FIGURAS

Figura 1. Distribución de reclamos por procesos. Fuente: Elaboración propia. 34
Figura 2. Cantidad de reclamos por procesos. Fuente: Elaboración propia. 36
Figura 3. Forma en como BNS calcula la relevancia de una palabra. Fuente: FORMAN, George. Choose your words carefully: An empirical study of feature selection metrics for text classification. En Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 2002. p. 150-162.....45
Figura 4. Estructura de la matriz de confusión. Fuente: Elaboración propia. 47
Figura 5. Matriz de confusión de ejemplo. Fuente: Elaboración propia. 48
Figura 6. Hiperplanos capaces de separar perfectamente los datos. Fuente: Elaboracion propia. 56
Figura 7. Maximización del margen hecho por las SVMs. Fuente: MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich.Introduction to information retrieval. Cambridge: Cambridge University Press, 2008, p 320. 56
Figura 8. Transformación del espacio mediante kernels. Izquierda: datos originales. Derecha: Datos transformados. Fuente: FLETCHER, Tristan. Support vector machines explained. Tutorial paper., Mar, 2009. 59
Figura 9. Conjunto de datos que no son linealmente separables. Fuente: Elaboracion propia. 61
Figura 10. Diagrama de construcción de los modelos. Fuente: Elaboración propia. 65
Figura 11. Medida F1 de los clasificadores con diferente número de features. Fuente: Elaboración propia. 68
Figura 12. Media de las desviaciones estándar de la medida F1 entre los 4 clasificadores para diferentes números de features. Fuente: Elaboración propia..68
Figura 13. Resultados de los clasificadores evaluados en cada proceso. Fuente: Elaboración propia. 73

LISTA DE TABLAS

Tabla 1. Procesos a utilizar para la construcción de los modelos.....33
Tabla 2. Distribución para los conjuntos de entrenamiento, validación y pruebas.35
Tabla 3. Tipos de kernel usados con SVM 60
Tabla 4. Resultados de los clasificadores con diferente representación de datos evaluados sobre el conjunto de validación. 66
Tabla 5. Parámetros para los clasificadores obtenidos usando validación cruzada sobre el conjunto de entrenamiento.....67
Tabla 6. Resultados finales usando Multinomial Naive Bayes.....69
Tabla 7. Resultados finales Usando Bernoulli Naive Bayes 70
Tabla 8. Resultados finales usando SVM con kernel lineal 70
Tabla 9. Resultados finales usando kernel gaussiano.....71

LISTA DE ECUACIONES

Ecuación 1. Función de mapeo a obtener mediante los clasificadores.....31
Ecuación 2. Normalización de features.....40
Ecuación 3. TF-IDF. Frecuencia de término - inversa de frecuencia de documento 40
Ecuación 4. Calculo de IDF (inversa de la frecuencia del documento) para TF-IDF 41
Ecuación 5. Normalización por coseno para TF-IDF 41
Ecuación 6. Calculo de BNS (Separación Bi-normal) 44
Ecuación 7. Calculo de precision y recall.....47
Ecuación 8. Medida F 49
Ecuación 9. Medida F1 49
Ecuación 10. Calculo de la medida F para clasificación entre múltiples clases.....49
Ecuación 11. Calculo de la clase de un documento usando el teorema de Bayes 51
Ecuación 12. Obtención de la expresión para el clasificador Naive Bayes a partir del teorema de Bayes 52
Ecuación 13. Calculo de la probabilidad a posteriori usando Naive Bayes Gaussiano.....53
Ecuación 14. Calculo de la probabilidad a posteriori usando Multinomial Naive Bayes.....53
Ecuación 15. Calculo de la probabilidad a posteriori mediante Bernoulli Naive Bayes.....54
Ecuación 16. Ecuación general de un hiperplano.....55
Ecuación 17. Ecuaciones de los márgenes que separan los datos 57
Ecuación 18. Calculo de la distancia entre los dos márgenes 57
Ecuación 19. Restricciones para la separacion de datos por margen 57
Ecuación 20. Función de minimización usada por la SVM 58

Ecuación 21. Función de minimización simplificada usada por las SVM 58
Ecuación 22. Función de minimización de la SVM usando margen suave 60

1. OBJETIVOS

1.1 OBJETIVO GENERAL

Construir un modelo de clasificación automática de quejas y reclamos para el sistema de gestión de calidad de la Universidad Tecnológica de Bolívar, utilizando Machine Learning.

1.2 OBJETIVOS ESPECÍFICOS

- Identificar de manera preliminar a través de una revisión literaria, las técnicas de Machine Learning y procesamiento de lenguaje natural - NLP más adecuadas para la clasificación automática de quejas y reclamos.
- Construir un conjunto de modelos utilizando las técnicas de Machine Learning y NLP identificadas, para realizar el proceso de clasificación automática de quejas en el sistema de apoyo a la gestión de la calidad de la Universidad Tecnológica de Bolívar, CalidadOnline.
- Construir un conjunto de datos destinados para el entrenamiento, validación y pruebas de los modelos construidos, con miras a la selección del más adecuado para la tarea propuesta.
- Evaluar los modelos y en base a los resultados determinar si existe uno apto para el funcionamiento con el actual sistema de apoyo a la gestión de la calidad de la Universidad Tecnológica de Bolívar.

2. PROBLEMA DE INVESTIGACIÓN

2.1 DESCRIPCIÓN DEL PROBLEMA

En la mayoría de las organizaciones no solo es suficiente con lograr los objetivos propuestos, también es relevante la forma como estos son alcanzados, es por esto que las organizaciones se encuentran en una constante búsqueda de mecanismos que les permitan mejorar los procesos utilizados en la consecución de los objetivos. Uno de estos procesos es el relacionado con el manejo de las quejas internas que se presentan en las empresas.

Todos los procesos que existen en una organización deben ser de una u otra forma evaluados con el fin de garantizar la calidad de los mismos y evaluar las oportunidades de mejoras y las necesidades de cambio de estos. Mediante la recolección de reclamos las organizaciones se aseguran de mantener la calidad de todos sus procesos y asegurar la continua conveniencia, adecuación y eficacia de estos. Es por eso que los procesos de gestión de reclamos y quejas juegan un papel importante en el funcionamiento de cualquier tipo de empresa, y la Universidad Tecnológica de Bolívar no es la excepción.

En la Universidad Tecnológica de Bolívar, en la actualidad el proceso de gestión de reclamos es realizado a través del sistema de apoyo a la calidad CalidadOnline. Un personal es encargado de recibir todos los reclamos de todos los procesos dentro de la universidad por medio de esta plataforma, la cual expone un formulario para el envío de estos (1). Los reclamos son revisados por el personal de la dirección de calidad y de acuerdo al área del reclamo seleccionado

en el formulario por parte del usuario, se es enviado al departamento encargado de implementar la solución. Esta implementación del proceso de gestión de reclamos funciona bien; sin embargo en la tarea de selección del área a la cual se enviara el reclamo, el usuario puede categorizarlo de forma incorrecta asignándole un área diferente al cual en realidad pertenece. Por tal motivo el personal de dirección de calidad se ve en la tediosa tarea de revisar semanalmente todos los reclamos recibidos con el fin de verificar que el área seleccionada en el formulario corresponda con el tipo de problema descrito en el reclamo.

Por todo lo expuesto anteriormente, se propone diseñar un modelo que pueda determinar de forma correcta el área que está relacionado a un reclamo, de tal forma de que se pueda realizar de forma automática la validación del área de todos los reclamos y no de forma manual por parte del personal de dirección de calidad de la universidad.

2.2 FORMULACIÓN DEL PROBLEMA.

Con base en lo expuesto anteriormente, se pretende dar solución al siguiente problema de investigación,

¿Cómo determinar de forma correcta y automática el proceso el cual está relacionado a un reclamo realizado en la Universidad Tecnológica de Bolívar?

3. JUSTIFICACIÓN

Las técnicas de Machine Learning cada vez más son de mayor utilidad en diversidad de tareas relacionado con la industria. Debido a su facilidad de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos, se han convertido en candidatas para realizar muchas tareas en las que la cantidad de errores que comete es menor en comparación con los humanos.

En la Universidad Tecnológica de Bolívar, el proceso de gestión de reclamos se realiza a través del sistema de apoyo a la calidad CalidadOnline. Los reclamos son recibidos a partir de un formulario en el cual el usuario selecciona el proceso al cual está relacionado el proceso. Al dejar al usuario la tarea de seleccionar el proceso del reclamo, se corre el riesgo de que este sea enviado a un departamento erróneo debido a que puede seleccionar un proceso diferente al que realmente pertenece. Por este motivo el personal de dirección de calidad se ve en la tediosa tarea de verificar semanalmente que el proceso seleccionado para cada queja sea el correspondiente.

Por tal motivo el diseño de un modelo de identificación del proceso relacionado a un reclamo o queja, permitiría, además de la automatización de esta tarea, facilitaría el trabajo del personal de gestión de calidad evitando la tarea de la revisión semanal de los reclamos recibidos en la Universidad Tecnológica de Bolívar.

4. ASPECTOS METODOLOGICOS.

4.1 TIPO DE INVESTIGACIÓN

Para la ejecución de la investigación, se utilizará un tipo de estudio de carácter descriptivo-correlacional. “La investigación descriptiva busca especificar, propiedades características y rasgos importantes de cualquier fenómeno que se analice mientras que los estudios correlacionarles tiene como propósito conocer la relación que exista entre dos o más conceptos, categorías o variables en un contexto en particular” (2).

En una etapa inicial el estudio es descriptivo porque se pretende ver en forma detallada las características del problema, que variables hay que tener en cuenta en la clasificación de quejas o reclamos, que tipos de modelos son los más adecuados para la resolución del problema, y los mecanismos implementados en estos; Posteriormente se implementará un tipo de estudio correlacional, porque con este se pretende establecer relaciones entre los diferentes mecanismos y parámetros implementados en los modelos en contraste con las soluciones obtenidas.

4.2 METODOLOGIA DE TRABAJO.

Para el cumplimiento de los objetivos se establecieron un conjunto de actividades que sumadas podrán dar con el cumplimiento de los objetivos propuestos. En este caso la primera a realizar consiste en realizar una revisión bibliográfica que permita identificar que modelos de Machine Learning y técnicas de procesamiento

del lenguaje natural son las más utilizadas para abordar el problema de clasificación de documentos.

Una vez culminada esta etapa, se construirá un conjunto de modelos con las técnicas encontradas en la revisión bibliográfica, por lo cual se tendrá en cuenta los diferentes métodos de extracción de características para la clasificación, el tipo de entrenamiento del modelo y la forma en que estos serán evaluados. Finalizada esta etapa, el siguiente paso consiste en la construcción de un conjunto de datos de entrenamiento que se utilizaran para entrenar el modelo de Machine Learning, un conjunto de datos de validación que se utilizaran para comparar el rendimiento de cada uno de los modelos y un conjunto de datos de prueba que se utilizaran para medir la eficacia del modelo seleccionado para la clasificación de quejas.

Con base a los resultados que se hayan obtenido, se determinara si existe un modelo con resultados lo suficientemente buenos como para poder trabajar en conjunto con el actual sistema de gestión de calidad de la Universidad.

5. MARCO REFERENCIAL

5.1 MARCO CONCEPTUAL

A continuación se presenta un conjunto de conceptos que servirán como base en el desarrollo de la presente investigación:

- **Reclamo**: protesta u oposición que se realiza contra algo.
- **Sistema de gestión de calidad:** forma de trabajo mediante la cual una organización asegura la satisfacción de las necesidades de sus clientes. Para la cual, planifica, mantiene y mejora continuamente el desempeño de sus procesos, bajo de un sistema de eficiencia y eficacia que le permite lograr ventajas competitivas.
- **Modelo:** Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento (3).
- **Aplicación:** Programa preparado para una utilización específica, como el pago de nóminas, formación de un banco de términos léxicos, etc. (4)
- **Clasificación:** Un problema de clasificación trata de tomar un conjunto de elementos y agruparlos por clases.
- **Generalizar:** Abstraer lo que es común y esencial a muchas cosas, para formar un concepto general que las comprenda todas (5).

5.2 MARCO TEORICO

5.2.1 Machine Learning.

Machine Learning es una rama de la inteligencia artificial interesada por el estudio de algoritmos computacionales que mejoran automáticamente a través de la experiencia (6). A partir de un conjunto de ejemplos suministrados (experiencia), el programa es entrenado con el objetivo de aprender a realizar una tarea sin ser programado explícitamente. La importancia de estas técnicas radica en que en algunos casos no es posible definir un algoritmo capaz de resolver un problema particular, por lo que es necesario que la maquina aprenda a resolverlo a partir de ejemplos que demuestran la forma en que lo debe hacer.

La creación de un sistema de Machine Learning involucra una serie de etapas que están directamente relacionadas con la eficacia de este: Selección de características, construcción del modelo de Machine Learning, aprendizaje y evaluación.

La selección de características es el proceso de identificación y eliminación de información irrelevante y redundante tanto como sea posible; esto reduce la dimensionalidad de los datos y puede permitir que los algoritmos de aprendizaje operen con mayor rapidez y eficacia (7).

Existe una gran variedad de algoritmos de Machine Learning. Entre los que usan entrenamiento supervisado se destacan: redes neuronales artificiales, máquinas de soporte vectorial, regresión logística, KNN, entre otros. Por otro lado algoritmos que utilizan aprendizaje no supervisado se encuentran: K-means, análisis de

componentes principales, redes neuronales artificiales, mapas auto-organizados, entre otros. Para el caso de algoritmos semi-supervisados los más usados son: separación de baja densidad, método de consistencia, modelo de campo aleatorio gaussiano, entre otros.

Las técnicas de Machine Learning son dividas de acuerdo al proceso de aprendizaje que el algoritmo utiliza: Aprendizaje supervisado, no supervisado y semi-supervisado.

En el aprendizaje supervisado, el objetivo es aprender una función que relacione un conjunto de entradas a una salida cuyos valores correctos son proporcionados por un supervisor, de tal forma que esta función también pueda generalizar su comportamiento para entradas no usadas en el entrenamiento (8). En el campo de Machine Learning generalizar se puede definir como: “la habilidad que tiene un algoritmo de realizar con precisión una tarea con datos no vistos después de ser entrenados en un conjunto de datos” (9).

En el aprendizaje no supervisado, no existe un supervisor como tal que guie el proceso de aprendizaje, sino que se ha previsto de una medida de la calidad de representación de los datos que el modelo requiere aprender, y los parámetros de este son optimizados con respecto a esa medida (10). Una vez que el modelo ha optimizado sus parámetros, este desarrolla la habilidad de formar representaciones internas de las características de los datos de entrada y por lo tanto crea nuevas clases automáticamente (11).

El aprendizaje semi-supervisado, como su nombre lo indica es una combinación de aprendizaje supervisado y no supervisado. Tradicionalmente clasificadores usan datos etiquetados para ser entrenados, sin embargo los datos etiquetados son difíciles de obtener, y toma un tiempo considerable conseguirlos. Por el otro lado los datos no etiquetados son relativamente fáciles de obtener pero hay pocas formas de utilizarlos. El entrenamiento semi-supervisado maneja este problema usando grandes cantidades de datos no etiquetados, junto con pequeñas cantidades de datos etiquetados para construir un mejor modelo (12).

Debido a que el propósito de la fase de aprendizaje es obtener conocimiento útil para la solución de una tarea, deben existir unos criterios con los que se pueda medir que tan bien nuestro modelo aprendió a resolverla (13). Esta evaluación se hace con un conjunto de datos diferente al utilizado en la fase de entrenamiento para ver qué tan bién el modelo generaliza al presentarle nuevas entradas. Existe diferentes técnicas para la evaluación de sistemas de Machine Learning, entre las que se destacan: precisión, recall, curva ROC, medida F, entre otros.

5.2.2 Procesamiento del lenguaje natural.

El Procesamiento del lenguaje natural involucra un conjunto de técnicas computacionales para el análisis y representación de textos de forma natural en uno o varios niveles de análisis lingüístico con el fin de lograr el desarrollo del procesamiento del lenguaje de la misma forma que los humanos para una variedad de tareas o aplicaciones (14).

Las técnicas de NLP (procesamiento del lenguaje natural en inglés), son divididas de acuerdo a siete niveles de análisis lingüístico para la extracción de la información (15):

- Fonético o fonológico, que se encarga de la pronunciación. En un sistema NLP que acepta como entrada un flujo de habla, las ondas de sonidos son analizadas y codificadas a una señal digital para la interpretación de diversas normas o para compararla con un particular modelo de lenguaje que se esté utilizando.
- Morfológico, que trata con las partes más pequeñas de las palabras que tienen un significado (morfemas). Dado que el significado de cada morfema sigue siendo el mismo a través de las palabras, los seres humanos pueden descomponer una palabra desconocida en sus morfemas constituyentes con el fin de entender su significado.
- Nivel léxico, que se ocupa del significado individual de las palabras.
- Nivel sintáctico, que se ocupa de la gramática y la estructura de las sentencias. La salida de este nivel de procesamiento es una representación de la oración que muestra las relaciones de dependencias estructurales entre las palabras. Para esto se requiere una gramática y un reconocedor de dicha gramática.
- Nivel semántico, que trata con el posible significado de una oración centrándose en los significados de las palabras que la constituyen. En este nivel de procesamiento se incluye la desambiguación semántica de palabras con múltiples sentidos.
- Nivel de discurso que trata con la estructura de diferentes clases de textos usando la estructura del documento. Mientras que en los niveles sintáctico y semántico se trabaja a nivel de oraciones, en este se enfocan en las propiedades del texto como un todo, que trasmiten un significado al hacer las conexiones entre las oraciones.
- Nivel pragmático que trata con el conocimiento que proviene del contexto más allá del contenido del texto. El objetivo es usar información del

contexto en la interpretación del significado del texto.

5.2.3 Clasificación de texto.

La clasificación de texto (también conocido como categorización de texto) es la tarea de automáticamente agrupar un conjunto de documentos entre un conjunto de categorías predefinidas. La clasificación de texto puede soportar y mejorar varias tareas tales como etiquetado automático por tópico(es decir la asignación de etiquetas a los documentos), creación de bibliotecas digitales, filtrado de spam, detección de contenido para adulto, plagio, mejora en la precisión de los buscadores web, e incluso ayudar a los usuarios a interactuar con los motores de búsqueda (16).

La clasificación de texto por lo general sigue una estrategia de aprendizaje supervisado, donde por primera vez se construye un modelo de clasificación de acuerdo con los documentos pre-clasificados y luego usar el modelo para clasificar nuevos documentos que no se han visto. La construcción de modelos de clasificación de texto por lo general significa encontrar el conjunto de características que mejor identifican las clases de documentos (17).

5.2.4 Sistema de gestión de la calidad ISO 9001.

La ISO 9001 es una norma internacional para un sistema de gestión de calidad que se centra en todos los elementos de administración de calidad con los que una empresa debe contar para tener un sistema efectivo que le permita administrar y mejorar la calidad de sus productos o servicios (18). Esta normal internacional promueve la adopción de un enfoque basado en procesos cuando se desarrolla, implementa, y mejora la eficacia de un sistema de gestión de la calidad,

para aumentar la satisfacción del cliente mediante el cumplimiento de sus requisitos.

La norma ISO 9001 está estructurada en 8 capítulos en los que se describen los requisitos para implementar cualquier sistema de gestión (19):

- Capítulo 1 al 3: Guías y descripciones generales.
- Capítulo 4, Sistema de gestión: Contiene los requisitos generales para gestionar la documentación.
- Capítulo 5, Responsabilidades de la dirección: Contiene los requisitos que debe cumplir la dirección de la organización, tales como definir la política, asegurar que las responsabilidades y autoridades están definidas, aprobar objetivos, el compromiso de la dirección con la calidad, etc.
- Capítulo 6, Gestión de los recursos: la Norma distingue 3 tipos de recursos sobre los cuales se debe actuar: RRHH, infraestructura, y ambiente de trabajo. Aquí se contienen los requisitos exigidos en su gestión.
- Capítulo 7, Realización del producto/servicio: aquí están contenidos los requisitos puramente de lo que se produce o brinda como servicio (la norma incluye servicio cuando denomina "producto"), desde la atención al cliente, hasta la entrega del producto o el servicio.
- Capítulo 8, Medición, análisis y mejora: aquí se sitúan los requisitos para los procesos que recopilan información, la analizan, y que actúan en consecuencia. El objetivo es mejorar continuamente la capacidad de la organización para suministrar productos y/o servicios que cumplan con los requisitos. El objetivo declarado en la Norma, es que la organización busque sin descanso la satisfacción del cliente a través del cumplimiento de los requisitos.

Haciendo énfasis en el módulo Medición, análisis y mejora, la organización debe realizar el seguimiento de la información relativa a la percepción del cliente con

respecto al cumplimiento de sus requisitos por parte de la organización. Uno de este tipo de información a seguir es la relacionada con los procesos en la organización, por lo que se debe aplicar métodos apropiados para el seguimiento y la medición de los procesos del sistema de gestión de calidad. Estos métodos deben demostrar la Capacidad de los procesos para alcanzar los resultados planificados.

Además la organización debe determinar, recopilar y analizar los datos apropiados para demostrar la idoneidad y la eficacia del sistema de gestión de la calidad. El análisis de datos debe proporcionar información sobre: La satisfacción del cliente, la conformidad con los requisitos del producto, las características y tendencias de los procesos y de los productos, y los proveedores.

Una vez establecidas políticas para la recolección de datos, se deben tomar acciones correctivas para eliminar las causas de las no conformidades con objeto de que no vuelvan a ocurrir. Se debe establecer un procedimiento documentado para definir los requisitos para revisar las no conformidades (incluyendo las quejas de los clientes), determinar las causas de las no conformidades, evaluar la necesidad de adoptar acciones para prevenir las no conformidades, determinar e implementar las acciones necesarias, registrar los resultados de las acciones tomadas y revisar la eficacia de las acciones correctivas tomadas.

6. ESTADO DEL ARTE

La clasificación de texto (también conocido como categorización de texto) es la tarea de automáticamente agrupar un conjunto de documentos entre un conjunto de categorías predefinidas. La clasificación de texto ha sido aplicada en diversidades de tareas, en las que se destacan: detección de spam en e-mails, determinación automática del genero de un texto, e incluso evaluación automática de ensayos; Sin embargo en los últimos años esta técnica ha jugado un papel importante en las organizaciones o empresas debido a la mayor disponibilidad de documentos en formato digital y la necesidad de organizarlos. Anteriormente esta tarea se hacía de forma manual por parte de un personal, pero esta solución dejo de ser viable por la gran cantidad de textos manejado por las grandes organizaciones. Es por esto que diversas técnicas para la automatización de estos procesos han surgido y han tomado gran aceptación por la comunidad científica.

Para dar solución al problema de clasificación de texto han existido dos tipos de técnicas. La primera de ellas, que fue popular en los 80's, fue la ingeniería del conocimiento (knowledge engineering), la cual consistía en la definición manual de un conjunto de reglas que codificaban el conocimiento sobre la forma de clasificar los documentos dentro las categorías dadas. Sin embargo Este enfoque fue perdiendo popularidad a favor de las técnicas de machine learning, las cuales construían automáticamente un clasificador de texto mediante el aprendizaje de un conjunto de documentos previamente clasificados. La ventaja de este enfoque es que se logra una precisión comparable a la lograda por personas expertas en la categorización de documentos, y un considerable ahorro en términos de personas expertas, debido a que no se necesitaba intervención de expertos en el dominio para la construcción del clasificador. A continuación se realizara una revisión de

las investigaciones previamente realizadas en este campo de investigación, buscando establecer bases teóricas para el presente proyecto de investigación.

En 1999 Larkey (20) desarrolla un sistema para la organización de documentos de patentes de EE.UU para hacer su búsqueda más sencilla, usando como clasificador el algoritmo k-nearest-neighbor. En 2002 Stijn Viaene y compañeros (21) realizan una investigación donde evaluaron varias técnicas de clasificación de texto aplicada a la detección de fraude en reclamos de seguros de automóvil, usando técnicas como regresión logística, arboles de decisión C4.5, k-nearest - neighbor, perceptron multicapa usando aprendizaje bayesiano, máquinas de soporte vectorial y clasificador bayesiano ingenuo (naive bayes).

En 2007 Kwon y Hovy (22) desarrollaron una herramienta de clasificación de comentarios recibidos por las agencias reguladoras de estados unidos al momento de anunciar proyectos de reglamentos. Los comentarios son clasificados en las siguientes categorías: apoya, en contra o propone una nueva idea, y el modelo fue aplicado a los comentarios del publico acerca de la propuesta de reglas estándares de emisión de contaminantes peligrosos, propuesta por la agencia de protección ambiental (EPA). Para la clasificación utilizan máquinas de soporte vectorial.

EN 2008 Yu, Kaufmann y Diermeir (23) discuten el diseño de un clasificador de partidos políticos para los discursos de los congresistas usando máquinas de soporte vectorial y naive bayes, y examinan estos clasificadores dependiendo de los autores de los discursos y de la época en que estos fueron hechos.

Un campo que ha tomado bastante interés en los últimos años en las organizaciones es la detección de fraude de diversos tipos (fraude bancario, fraude de seguros, fraude de valores y materia prima, entre otros). En 2011 Ngai y compañeros (24) realizan una revisión de técnicas de minería de datos aplicadas a la detección de fraude financiero de diverso tipo, aplicando técnicas como regresión logística, redes neuronales y árboles de decisión.

Hasta ahora se han presentado diferentes investigaciones de clasificación de texto aplicado en empresas u organizaciones, todos estos usando técnicas de Machine Learning. Como se mencionó antes, este tipo de técnicas son actualmente las más usadas para la tarea de clasificación de texto, sin embargo existe otros tipo de técnicas con las que se ha podido lograr resultados comparables con los logrados utilizando Machine Learning. A continuación se expone investigaciones donde se utilizan este tipo de técnicas.

En 1994 Hock (25) presente el sistema INFOCLAS en el cual se aplican métodos estadísticos de information retrieval y fuentes de conocimientos especificas del dominio del problema para la clasificación de cartas de negocio alemanas dentro de su correspondiente tipo de mensaje.

De igual forma la clasificación de texto se ha aplicado en el ámbito médico. En 2008 Perea (26) presenta un sistema de categorización automática de expedientes de niños con enfermedades respiratorias, usando las ontologías del metatesauro UMLS (Unified Medical Language System). En 2006 Maña (27) utiliza técnicas de categorización automática de textos basadas en la utilización de ontologías y recursos léxicos para el acceso a información bilingüe en el ámbito biomédico.

En 1997 Chris Clack y compañeros (28) usaron técnicas de programación genética para desarrollar un sistema para la organización ambiental FOE (Friends of the Earth), el cual consistía en la clasificación de documentos de todo tipo (emails, web pages, Usenet etc) y que estos fueran enviados hacia los grupos de investigación interesados dependiendo del tópico del documento. De igual forma Yolis (29) utilizó Algoritmos genéricos introduciendo 5 nuevos operadores, diseñados específicamente para la resolución del problema de categorización.

Otra aplicación de la clasificación de texto es el filtrado de texto (text filtering), la cual consiste en clasificar un flujo de documentos de entrada enviados de forma asíncrona por un productor de información a un consumidor de información. Un típico caso de filtrado de texto es un productor de noticias, expuesto en 1990 por Hayes (30). En este caso el sistema de filtro deberá bloquear la entrega de noticias a los consumidores que probablemente no les interesen (por ejemplo noticias diferentes de deportes deberán ser bloqueadas para un periódico de deportes). Para esto se construye un clasificador basado en reglas las cuales fueron definidas de forma manual. De igual forma en 2011 Lana y compañeros (31) utilizan un clasificador basado en reglas pero estas son generadas de forma automática por un algoritmo de aprendizaje computacional (KNN) con el fin de crear un método hibrido de categorización automática de texto.

7. DEFINICIÓN FORMAL DEL PROBLEMA.

Dado un reclamo el cual es representado como una sucesión de palabras $r_i = \{t_0, t_1, t_2, \dots, t_n\} \in X$, donde r_i es el reclamo y t_k es la palabra en la posición k en el reclamo; y sea $P = \{p_0, p_1, p_2, \dots, p_k\}$ el conjunto de procesos y un conjunto de reclamos etiquetados (llamado datos de entrenamiento) $D = \langle r_1, p_1 \rangle, \langle r_2, p_2 \rangle, \dots, \langle r_l, p_l \rangle$ donde $\langle r_i, p_i \rangle \in X \times P$, se desea obtener un clasificador o una función de clasificación γ que mapee reclamos a sus correspondientes procesos:

$$\gamma : X \underset{D}{\rightarrow} P$$

Ecuación 1. Función de mapeo a obtener mediante los clasificadores

El método de aprendizaje τ toma los datos de entrenamiento D como entrada y devuelve la función de clasificación aprendida γ. Este tipo de aprendizaje es llamado aprendizaje supervisado debido a que un supervisor (la persona que define las categorías y etiqueta los datos) sirve como un profesor guiando el proceso de aprendizaje.

8. CONSTRUCCIÓN DE CONJUNTOS DE DATOS

Para el funcionamiento de los modelos de Machine Learning, se es necesario tener un conjunto de datos los cuales van a ser la base de todos los modelos y a partir del cual se obtendrá toda la información necesaria para entrenarlos en la tarea de clasificación de reclamos. Por esta razón esta etapa es fundamental para la construcción de los clasificadores.

El área de gestión de calidad de la Universidad Tecnológica de Bolívar trabaja en conjunto con el sistema “CalidadOnline”, el cual de entre todas sus funciones tiene la responsabilidad de recibir los reclamos o quejas realizado por cualquier persona involucrada con la universidad. Estos reclamos son registrados en una base de datos y pueden ser exportados en cualquier formato para su manipulación. En la petición de los datos para realizar el presenta trabajo de investigación, se nos entregaron todas los reclamos hechos en la Universidad desde que el sistema fue puesto en marcha en un formato de Excel. A continuación se describirán las características de los datos recogidos por el sistema durante los 4 años que lleva en funcionamiento.

8.1 CATEGORIZACIÓN DE LOS RECLAMOS

Actualmente los reclamos son categorizados en 43 procesos, lo que indicaría que los modelos deben aprender a separar los reclamos en 43 grupos. Sin embargo el número de reclamos de la mayoría de las categorías es muy bajo para hacer que los modelos aprendan a determinar con alta precisión el proceso al cual pertenece. Por este motivo se es necesario trabajar solamente con categorías que

tengan un número suficiente de reclamos para poder entrenar correctamente un clasificador. En nuestro caso hemos establecido un umbral de 30 reclamos por clases. Esto quiere decir que las categorías que tengan un número de reclamos menor a 30 no serán tomadas en cuenta.

Categorías	Frecuencia
Mantenimiento planta física	97,00
Servicios administración financiera	61,00
Servicios Informáticos	44,00
Bienestar Universitario	31,00
Servicios Bibliotecarios	161,00
Servicios Académicos	31,00
Medición, Análisis y Mejora	57,00

Tabla 1. Procesos a utilizar para la construcción de los modelos

La siguiente grafica muestra cómo queda compuesto el conjunto de datos con los datos a utilizar para construir los modelos:

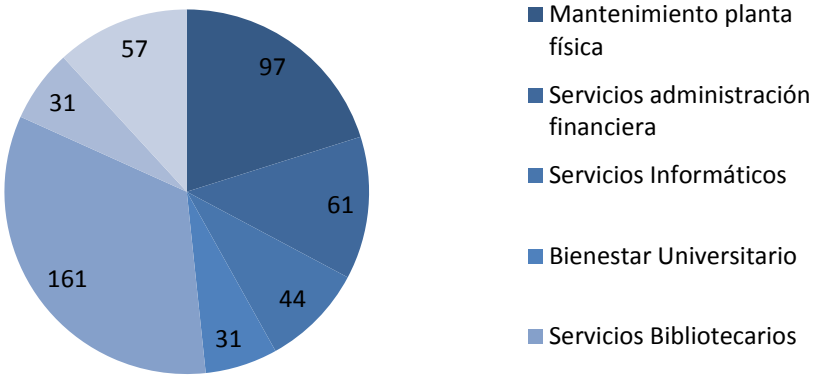

Figura 1. Distribución de reclamos por procesos. Fuente: Elaboración propia.

8.2 DIVISIÓN DEL CONJUNTO DE DATOS.

Para la construcción de los clasificadores es necesario en primera instancia dividir el conjunto de datos en dos grupos: Datos de entrenamiento y de pruebas. Los datos de entrenamiento son utilizados para la construcción de los modelos. De este conjunto de datos es que los clasificadores obtienen toda la información necesaria para la aprender a clasificador los reclamos. Por otra parte los datos de prueba son utilizados para medir que también los clasificadores realizan su tarea.

Por lo general cualquier clasificador tiene un conjunto de parámetros que rigen la forma en como estos funcionan. Variando estos parámetros se obtienen diferentes resultados en el proceso de clasificación por lo que es necesario obtener los parámetros que mejoren el resultado de la clasificación. Por tal motivo se es necesario tener otro conjunto de datos - datos de validación - que serán utilizados para probar los modelos con diferentes valores en sus parámetros y así

determinar con cuales parámetros se obtienen mejores resultados. Este paso puede involucrar heurísticas para la optimización de los parámetros.

Con base en todo lo anterior, Los datos serán divididos en tres grupos: entrenamiento, validación y pruebas. Con los siguientes tamaños para cada grupo:

Conjunto de dato	Tamaño
Entrenamiento	60%
Validación	10%
Pruebas	30%

Tabla 2. Distribución para los conjuntos de entrenamiento, validación y pruebas

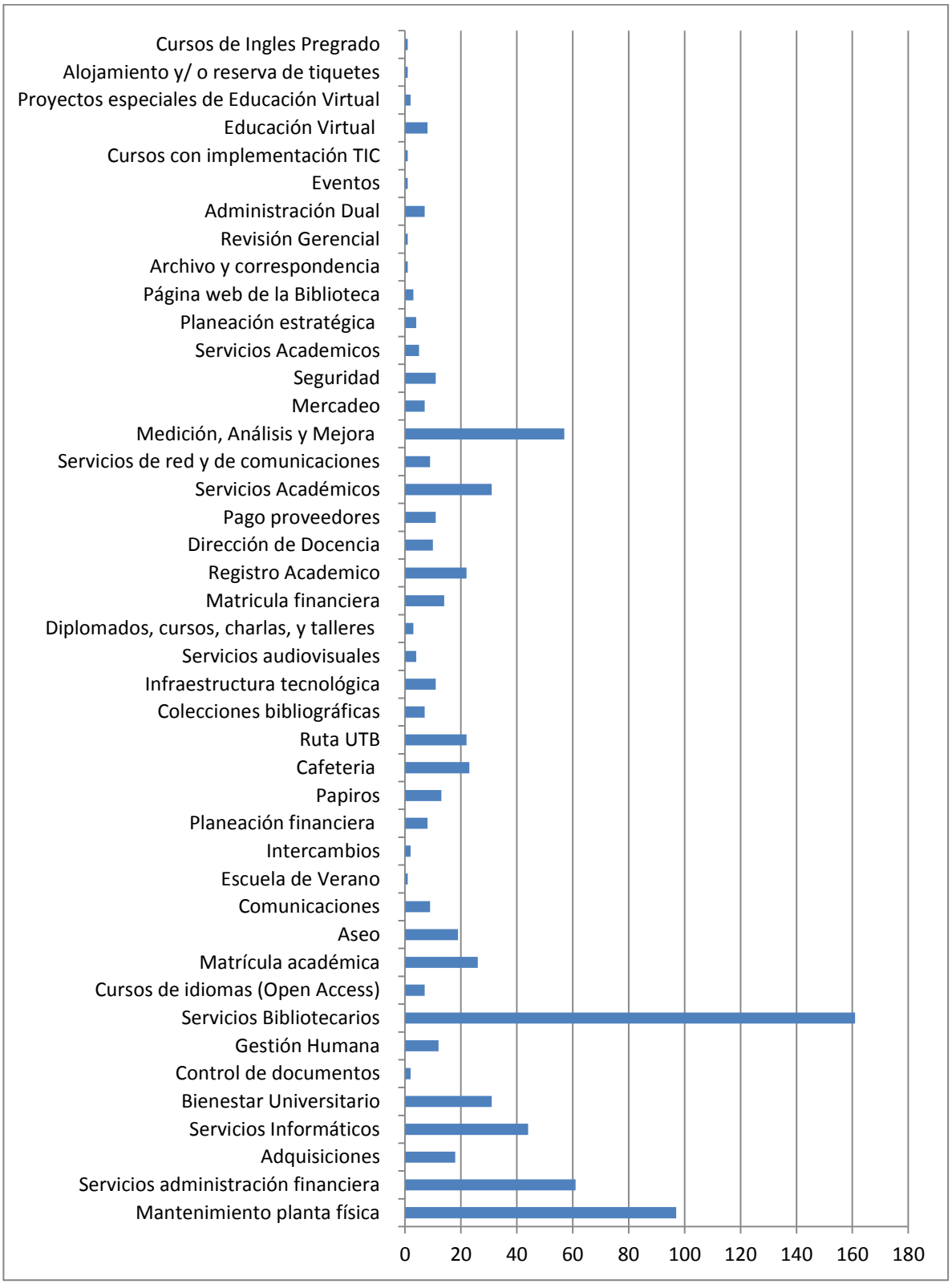

Figura 2. Cantidad de reclamos por procesos. Fuente: Elaboración propia.

9. PROCESAMIENTO DE TEXTO.

Por lo general todo tipo de texto trae consigo aparte de su información elementos utilizados para mejorar la lectura de las personas, como signos de puntuación, guiones, tildes, caracteres no alfa-numéricos, etc. Estos símbolos son útiles para las personas pero para los clasificadores es irrelevante y puede causar problemas al encontrar similitudes entre palabras. Es por eso que antes de que las quejas sean usadas para entrenar los clasificadores es necesario realizar un proceso de limpiado y normalizado a los datos con el fin de eliminar todos estos símbolos.

Además existen un conjunto de palabras cuya presencia en el corpus no aporta información al proceso de clasificación, estas palabras son denominadas palabras vacías (stop words). Palabras como artículos, pronombres, preposiciones, etc., son consideradas palabras vacías porque no tienen ningún significado y su eliminación mejoraría el rendimiento al procesar el corpus sin causar perdida de información sensible.

Por último es necesario aplicar un proceso conocido como stemming, el cual consiste en llevar cada palabra a su raíz con el fin de agrupar palabras con significado similar. Eso quiere decir que palabras como biblioteca y bibliotecario serán tratadas iguales debido a que su raíz es la misma (bibliotec). Para la conversión de cada palabra a su raíz se usa una implementación del algoritmo de snowball (32)

10. VECTORIZACIÓN DE LOS RECLAMOS.

Los textos no pueden ser directamente interpretados por un clasificador. Es por esto que deben ser convertidos en una representación vectorial de features[1], mediante un proceso de indexado que mapee cualquier texto a una representación compacta de su contenido. Existen diferentes formas de representar un texto en vectores, en nuestro caso se utilizaran dos: bag of words y TF-IDF.

10.1 BAG OF WORDS.

Es una representación simplificada usada en tareas relacionadas con el procesamiento de lenguaje natural. Consiste en crear una lista del vocabulario que se encuentre en todo el corpus. Creado el vocabulario, la representación del texto va a ser un vector que represente la frecuencia de cada palabra del vocabulario en el texto a transformar, de esta forma cada frecuencia de cada palabra en cada texto va a ser usada como feature para entrenar los clasificadores. Así suponiendo que el corpus este conformado por los siguientes dos reclamos:

- *Se están presentando filtraciones de agua lluvia en las salas de malocka y poeta.*
- *Filtraciones de agua lluvia en oficina de investigaciones y laboratorio de ondas.*

[1] Feature: Especificación de una propiedad individual que describe una instancia o ejemplo del problema. Por ejemplos para tareas de diagnóstico de enfermedades los síntomas de los pacientes representarían los features para cada paciente.

Basados en los dos anteriores textos, El vocabulario es:

$\{se, estan, presentando, filtraciones, de, agua, lluvia, en, las\ , salas, malocka, oficinas, investigaciones, y, laboratorio, ondas\}$

El cual tiene 15 diferentes palabras. Y usando la posición de cada palabra en la lista de vocabulario, cada reclamo es representado por un vector de 16 elementos:

$$[1,1,1,1,1,1,1,1,1,1,0,0,0,0,0]$$
$$[0,0,0,1,2,1,1,1,0,0,0,1,1,1,1,1]$$

Cabe resaltar que en el anterior ejemplo a los reclamos no se les ha aplicado el procesamiento de texto, por lo que las palabras: “se”, “están”, “de”, “en”, “las” y “y”, son eliminadas del corpus y el resto de palabras son llevadas a su raíz.

En algunos casos algunas palabras pueden aparecer numerosas veces en comparación con otras lo cual puede causar problemas en la función objetivo de algunos clasificadores. Esto se debe básicamente a que al calcular la distancia entre dos vectores, si uno de los features tiene un amplio rango de valores, la distancia estará gobernada por ese feature. Por eso es altamente recomendable normalizar el rango de cada uno de los features de tal forma que cada feature contribuya proporcionalmente a la distancia final. Para la normalización de cada feature se usa la siguiente expresión:

$$x' = \frac{x - \mu}{\sigma}$$

Ecuación 2. Normalización de features

Donde x es el valor, x' es el valor normalizado, μ y σ es la media y la desviación estándar de el feature al que x pertenece.

10.2 TF-IDF.

Term Frecuency Inverse Document Frecuency es un método de ranking de palabras en el campo de procesamiento del lenguaje natural, el cual consiste en asignar a cada palabra un valor que refleja que tan importante es en el corpus; luego este valor es usado como feature para entrenar los clasificadores (33). Un valor alto TF-IDF implica una fuerte relación entre la palabra y los documentos en los que este aparece. TF-IDF es el producto de dos valores estadísticos, la frecuencia de la palabra y la inversa de la frecuencia del documento (ecuación 3), donde $TF(t,d)$ es la frecuencia de la palabra t en el documento d, y $IDF(t,D)$, medida de si el termino es común o raro a través de todos los documentos, es calculada usando la ecuación 4, donde $|A|$ representa la cardinalidad del conjunto A.

$$TF - IDF(t,d,D) = TF(t,d) * IDF(t,D)$$

Ecuación 3. TF-IDF. Frecuencia de término - inversa de frecuencia de documento

$$IDF(t, D) = \log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

Ecuación 4. Calculo de IDF (inversa de la frecuencia del documento) para TF-IDF

En otras palabras TF-IDF asigna al término t un valor en el documento d que es:

- Alto cuando el termino ocurre muchas veces dentro de un pequeño número de documentos (dando así gran poder de discriminación a los documentos).
- Bajo cuando el término ocurre pocas veces en un documento, u ocurre en muchos documentos (ofreciendo una señal de relevancia menos pronunciada).
- Bajo cuando el termino ocurre en virtualmente todos los documentos.

Con el fin de que los pesos de cada termino este en el intervalo $[0,1]$ y los documento sean representados como vectores de la misma longitud, los valores obtenidos con TF-IDF son normalizados usando la normalización por coseno (ecuación 5).

$$\frac{TF - IDF(t, d, D)}{\sqrt{\sum_i^{|V|} TF - IDF(t_i, d, D)^2}}$$

Ecuación 5. Normalización por coseno para TF-IDF

11. FEATURE SELECTION.

Es un proceso de selección de un subconjunto de los features original de tal forma que el espacio de características es óptimamente reducido de acuerdo a ciertos criterios de evaluación (34). Feature selection es frecuentemente usado como un paso de pre-procesamiento para los algoritmos de machine learning en áreas de aplicación en donde los datos están compuesto por decenas o cientos de miles de variables. Ejemplos de estas áreas son: análisis de expresión genética, química combinatoria y procesamiento de texto. Los beneficios que conlleva este proceso son varios: facilidad de visualización y mejor entendimiento de los datos, reducción en los requerimientos de almacenamiento, reducción en los tiempos de entrenamiento, y mejoras en el rendimiento de clasificación de los modelos.

La elección del algoritmo de feature selection se ha hecho en base a cómo controlar el problema de desbalance de datos entre las clases. Como se puede observar en la figura 1 la distribución de datos entre las 7 clases nos es equitativa, esto conlleva a cierto problema en la clasificación: Las clases con mayor número de documentos gobiernan el proceso de clasificación, haciendo que cualquier clasificador trivial ignore por completo las clases minoritarias. Para abordar este problema se usa un método iterativo planteado por Forman (35). El método consiste en realizar feature selección por cada clase separadamente vía descomposición binaria, y luego determinar el conjunto final de features mediante un algoritmo de planificación donde cada clase nomina sus features más destacados en cada iteración. El algoritmo es descrito a continuación.

```
subconjunto_features ← featureSelection[R, M](dataset)
    Para cada clase c del dataset:
        Asignar pesos a todos los features de acuerdo a M para la sub-tarea binaria
        de discriminar clase c  vs el resto de clases combinadas.
        Almacenar los pesos asignados a los features para cada clase c.
    Mientras subconjunto_features no esté completo:
        Llamar planificador R para seleccionar una clase c.
        Seleccionar la característica con el mayor peso generado por M para c.
        Agregar a subconjunto_features, si ya no está presente.
```

El primer parámetro R, es una política de planificación dinámica entre las clases. Para este parámetro se utiliza la política **Rand-Roubin**, el cual es un planificador aleatorio. Selecciona la siguiente clase aleatoriamente con probabilidad de acuerdo a la distribución de las clases. Para esto a cada clase se le asigna una probabilidad inversamente proporcional a la cantidad de documentos que contiene. De esta forma las clases con menor cantidad de datos (minoritarias), tendrán mayor probabilidad de ser seleccionadas y por lo tanto el sub-conjunto de features nuevo estará conformado en su mayoría por los features más relevantes de las clases minoritarias.

El segundo parámetro M, es un método de ranking de features para dos clases el cual asigna valores a features dependiendo de qué tan importante es esa feature para la discriminación de una clase con el resto. Basándonos en los resultados obtenidos por Forman (36), se ha decidido utilizar el método de Separación Bi-normal (37) debido a que genera los mejores resultados para la medida F en comparación con otros métodos de feature selection para clasificación de texto como: Information Gain, Chi-Squared, Odds Ratio, etc.

11.1 SEPARACION BI-NORMAL.

Separación Bi-normal (BNS) es una nueva medida de ranking de feature. Definida en la ecuación 6, donde F^{-1} es la distribución normal estándar de la función inversa de probabilidad acumulativa. BNS modela la ocurrencia de una palabra en cada documento, mediante el evento de una variable normal aleatoria excediendo un hipotético umbral. La tasa de prevalencia de la palabra corresponde al área bajo la curva después del umbral. Si la palabra es más frecuente en la clase positiva, entonces su umbral está más lejos del extremo de la distribución que el de la clase negativa, figura 3. Por lo que BNS mide la distancia entre estos dos umbrales.

$$|F^{-1}(tpr) - F^{-1}(fpr)|$$

$$tpr = \frac{numero\ de\ documentos\ positivos\ conteniendo\ la\ palabra}{total\ de\ documentos\ positivos}$$

$$fpr = \frac{numero\ de\ documentos\ falsos\ conteniendo\ la\ palabra}{total\ de\ documentos\ falsos}$$

Ecuación 6. Calculo de BNS (Separación Bi-normal)

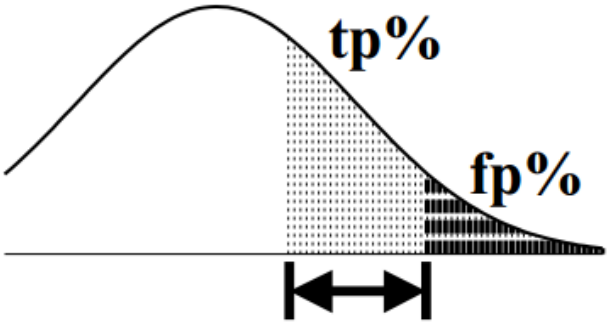

Figura 3. Forma en como BNS calcula la relevancia de una palabra. Fuente: FORMAN, George. Choose your words carefully: An empirical study of feature selection metrics for text classification. En Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 2002. p. 150-162.

Para el cálculo de la distribución normal estándar de la función inversa de probabilidad acumulativa se utiliza una aproximación minimax por funciones racionales implementada por Acklam (38).

12. METODOS DE EVALUACIÓN DE LOS CLASIFICADORES.

La evaluación de los modelos de clasificación de texto es realizada de forma experimentan, en vez de analítica. La razón de esto es que, para evaluar un sistema analíticamente (verificar que el sistema es correcto y completo) se es necesario una especificación formal del problema que el sistema está tratando de resolver, y la idea central de la clasificación de texto es, por su carácter subjetivo, inherentemente no formalizable. La evaluación experimental de un clasificador usualmente mide su efectividad, su habilidad para tomar la correcta decisión de clasificación.

La forma más sencilla de evaluar la efectividad de un clasificador es calculando el porcentaje de ejemplos clasificados correctamente sobre el total de documentos a clasificar. Sin embargo esta medida presente problemas cuando las clases de los documentos sufren de datos desbalanceados. Supongamos que se desea testear la efectividad de un clasificador binario para las clases “A” y “B”, para lo cual se construye un conjunto de datos de prueba con 10 datos para la clase “A” y 90 datos para la clase “B”. Suponiendo que se construya un simple algoritmo que siempre devuelva “B” para cualquier dato, la eficacia de nuestro clasificador será increíblemente de 90% a pesar de ser un algoritmo que no hace ningún tipo de procesamiento para determinar la clase del dato a clasificar. Es por esto que se utiliza medidas especiales como precisión, recall y F1 para determinar que también nuestro clasificador funciona.

12.1 PRECISION – RECALL.

En un problema de clasificación binaria (dos categorías), un clasificador etiqueta ejemplos como positivos o negativos. La decisión hecha por el clasificador puede ser representada en una estructura conocida como matriz de confusión o tabla de contingencia (39), figura 4. La matriz de confusión está conformada por 4 regiones: verdaderos positivos (TP) son datos correctamente clasificados como positivos, falsos positivos (FP) se refiere a datos negativos incorrectamente clasificados como positivo, verdaderos negativos (TN) corresponde a negativos correctamente clasificados como negativos, y finalmente falsos negativos (FN) se refiere a ejemplos positivos incorrectamente clasificados como negativos. A partir de estos valores se define dos medidas de rendimiento para los clasificadores: precision (P) y recall (R).

		Predicción	
		+	-
Categorías	+	TP	FN
	-	FP	TN

Figura 4. Estructura de la matriz de confusión. Fuente: Elaboración propia.

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

Ecuación 7. Calculo de precision y recall

De esta manera precisión es definido como la probabilidad de que si un documento aleatorio es clasificado como positivo, esta decisión sea correcta. Por otra parte recall se define como la probabilidad de que si un documento que debe

ser clasificado como positivo, esta decisión se tome. Cabe anotar que la clase que nos referimos como positivo siempre es la que presenta dificultad de detectar en la clasificación (clase minoritaria). Así usando estas dos medidas para el problema de clasificación entre las clases “A” y “B”, calculada la matriz de confusión (figura 5), se obtienen los valores de precisión y recall los cuales ambos van a ser cero.

		Predicción	
		A	B
categorías	A	0	0
	B	10	90

Figura 5. Matriz de confusión de ejemplo. Fuente: Elaboración propia.

12.2 MEDIDA F

La medida F, es una forma de evaluar el rendimiento de un clasificador unificando las medidas precision y recall para obtener un solo valor que represente la efectividad del modelo a probar (40). La fórmula general para la medida F es definida en la ecuación 8. Donde el parámetro β es un valor de peso que define que tan importante es precision en comparación con recall. Un valor mayor para β significa que el valor de precision será tomado más en cuenta que recall. Por lo general se usa el valor $\beta = 1$ para asignar igual importancia a ambas medida definiendo de esta forma la medida F1 (ecuación 9) a utilizar para la evaluación de los modelos.

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Ecuación 8. Medida F

$$F_1 = 2 \frac{precision \cdot recall}{precision + recall}$$

Ecuación 9. Medida F1

La medida F solamente puede ser aplicada para problemas de clasificación binaria. Para casos en que existan más de dos clases, se calcula la medida F para cada clase, donde la clase positiva es representada por la clase a calcular la medida F y la negativa representada por la unión de los datos de las demás clases. Y a partir de cada medida F se calcula una global que refleja la efectividad del clasificador. Existen diversas formas de calcular la medida F de los clasificadores, en nuestro caso se tomara el promedio ponderado por datos de pruebas con el fin de tener en cuenta el desbalance de datos en la evaluación. Esto quiere decir que dado n clases en donde F_i representa la medida F de la clase i, y sea l_i el número de ejemplos de la clase i en el conjunto de pruebas, la medida F global del clasificador está dada por la ecuación

$$F_{global} = \frac{\sum_i^n F_i * l_i}{\sum_i^n l_i}$$

Ecuación 10. Calculo de la medida F para clasificación entre múltiples clases

13. CONSTRUCCIÓN DE LOS MODELOS DE MACHINE LEARNING.

13.1 DESCRIPCIÓN DE LOS CLASIFICADORES DE MACHINE LEARNING.

Para la clasificación de texto se seleccionaran un conjunto de clasificadores de Machine Learning basados en su rendimiento en este tipo de problemas. Basándonos en el estado del arte, los clasificadores más idóneos y de mayor utilización en la tarea de categorización de texto han sido Naive Bayes y máquinas de soporte vectorial (SVM) por lo que serán incluidos estos algoritmos para la solución del problema. A continuación se explicara las generalidades de estos tres algoritmos a utilizar y sus variantes a la hora de implementarlos.

13.1.1 Naive Bayes.

Naive Bayes (también llamado clasificador bayesiano ingenuo) es un método probabilístico de aprendizaje supervisado basado en el teorema de Bayes (41) y la aplicación del principio de suposición de independencia. Dado un conjunto de clases $C = \{c_1, c_2, c_3, \dots, c_k\}$ supongamos que se desea determinar la clase c_d a la que pertenece el documento d. Para determinar la clase del documento se procede a calcular la probabilidad condicional de cada clase dado d, usando el teorema de Bayes, y se escoge la clase cuya probabilidad sea mayor, es decir c_d esta dado por la ecuación 11.

$$c_d = arg\max_{c \in C} P(c|d)$$

$$c_d = arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

Ecuación 11. Calculo de la clase de un documento usando el teorema de Bayes

Debido a que para el cálculo de la probabilidad de cada clase, $P(d)$ va a ser la misma, se puede remover de la ecuación 11.2 obteniendo la ecuación 12.1. Ahora, cualquier documento d está compuesto por una sucesión de palabras o tokens $\langle t_1, t_2, t_3, \dots, t_{n_d} \rangle$ por lo que la expresión para el cálculo de la clase es expresada en 12.2 la cual es la versión generalizada del clasificador bayesiano. Con esta última expresión hay un inconveniente al calcular $P(t_1, t_2, t_3, \dots, t_{n_d}|c)$ y es que hay muchos parámetros para calcular la probabilidad y además se deben calcular para cada clase, por lo que su complejidad computacional es bastante alta ($O(|x|^{n_d} \cdot |C|)$). Además es necesario tener una gran cantidad de datos de entrenamiento para poder calcular esta probabilidad y por lo general no se dispone de esa cantidad de datos. Es por eso que se hacen dos suposiciones para simplificar el proceso de clasificación. La primera es que no se tomara en cuenta la posición de las palabras en el documento, solo se tomara en cuenta que palabras ocurren en el documento. Y la segunda suposición es que cada palabra en el documento es estadísticamente independiente (42) de las otras, esto quiere decir que la presencia o ausencia de una palabra no depende de la existencia de otras palabras en el documento. Usando estas dos suposiciones se puede establecer la relación 12.3 y obteniendo en la ecuación 12.4 la expresión final para el cálculo de la clase de un documento mediante un clasificador bayesiano.

$$c_d \propto arg\max_{c \in C} \; P(d|c)P(c) \tag{12.1}$$

$$c_d \propto arg\max_{c \in C} \; P(c)P\left(t_1, t_2, t_3, \dots, t_{n_d}|c\right) \tag{12.2}$$

$$P\left(t_1, t_2, t_3, \dots, t_{n_d}|c\right) = P(t_1|c)P(t_2|c)\, P(t_3|c) \dots P\left(t_{n_d}|c\right) = \prod_{i=1}^{n_d} P(t_i|c)$$

$$c_d \propto arg\max_{c \in C} \; P(c) \prod_{i=1}^{n_d} P(t_i|c)$$

Ecuación 12. Obtención de la expresión para el clasificador Naive Bayes a partir del teorema de Bayes

Hay que aclarar que las dos suposiciones que se hicieron anteriormente son completamente erróneas sin embargo hace posible resolver el problema de clasificación de manera más fácil y con un alto grado de precisión a pesar de la simplificaciones hechas.

Como se muestra en la ecuación 12, para el uso del clasificador bayesiano es necesario calcular los valores $P(c)$ y $P(t_i|c)$ conocidos como “probabilidad a priori” y “probabilidad a posteriori” respectivamente. La probabilidad a priori es calculada dividiendo el número de documentos que pertenece a la clase c entre el total de documentos. Para la probabilidad a posteriori, existen diversas formas de calcularla. A continuación se expone las tres más usadas.

Naive Bayes Gaussiano.

Naive Bayes Gausiasano (43) usa la distribución gaussiana para estimar la probabilidad a posteriori, ecuación 13. Donde μ_c y σ_c son la media y la desviación estándar de la distribución. Estos valores son estimados usando maximum likelihood.

$$p(t_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{(-\frac{(t_i-\mu_c)^2}{2\pi\sigma_c^2})}$$

Ecuación 13. Calculo de la probabilidad a posteriori usando Naive Bayes Gaussiano

Multinomial Naive Bayes.

Multinomial Naive Bayes es una de las dos clásicas variantes de Naive Bayes utilizadas en clasificación de texto (en donde los datos son típicamente representados como vectores de frecuencia de palabras). Para el cálculo de la probabilidad a posteriori se utiliza la ecuación 14.

$$p(t_i|c) = \frac{N_{ic} + \alpha}{N_c + \alpha n}$$

Ecuación 14. Calculo de la probabilidad a posteriori usando Multinomial Naive Bayes

Donde $N_{ic} = \sum_{x \in T | y=c} t_i$ es el número de veces que el feature x_i aparece en documentos de la clase c en el conjunto de entrenamiento, y $N_c = \sum_{i=1}^{|T|} N_{yi}$ es la suma de la frecuencia de todos los features de la clase y. El valor α, conocido como parámetro de suavizado (smoothing), cuenta para las palabras no presentes en el conjunto de entrenamiento y previene que el valor de la probabilidad sea cero en el cálculo para cada clase. Estableciendo el valor de $\alpha = 1$ es conocido como Laplace smoothing (44), mientras que $\alpha < 1$ es llamado Lidstone smoothing.

Bernoulli Naive Bayes.

Bernoulli Naive Bayes es la otra variante de Naive Bayes más utilizada en la clasificación texto. En este método los datos son distribuidos de acuerdo a la distribución de Bernoulli multivariada, la cual genera un indicador para cada término del vocabulario, siendo 1 indicando la presencia o 0 la ausencia del termino en un texto. La forma en cómo se calcula la probabilidad a posteriori es mediante la ecuación 15.

$$p(t_i|c) = P(i|c)t_i(1 - P(i|c))(1 - t_i)$$

Ecuación 15. Calculo de la probabilidad a posteriori mediante Bernoulli Naive Bayes

Este método difiere de Multinomial Naive Bayes en que explícitamente penaliza la no ocurrencia de la palabra i que es un indicador para la clase c, mientras que el otro simplemente ignoraría la no ocurrencia de la palabra.

13.1.2 Máquinas de Soporte Vectorial.

Las máquinas de soporte vectorial (SVM) es un método de clasificación supervisado introducido en 1992 por Boser, Guyon and Vapnik (45). Su funcionamiento radica en la maximización de la distancia entre la frontera de clasificación y los datos. Considere el problema de clasificar el conjunto de datos de entrenamiento perteneciente a dos clases:

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^l, y^l)\}, \quad x \in R^n, \qquad y \in \{-1,1\}$$

Suponiendo que los datos son linealmente separables, existen infinito número de híper-planos capaces de dividir el conjunto de datos en las dos clases, figura 2. En este aspecto es donde radica la esencia de las SVM ya que buscan la mejor frontera de clasificación que separe los datos. Las SVM definen el hiperplano que mejor separa el conjunto de datos como el que maximiza la distancia (margen) entre el hiperplano de clasificación y el conjunto de vector más cercano a este, como se muestra en la figura 3. Supongamos que el hiperplano que separa los datos está dado por la siguiente expresión:

$$w \cdot x + b = 0$$

Ecuación 16. Ecuación general de un hiperplano

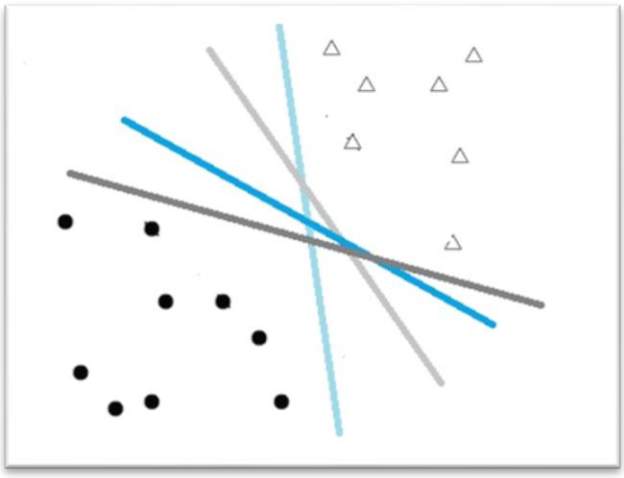

Figura 6. Hiperplanos capaces de separar perfectamente los datos. Fuente: Elaboracion propia.

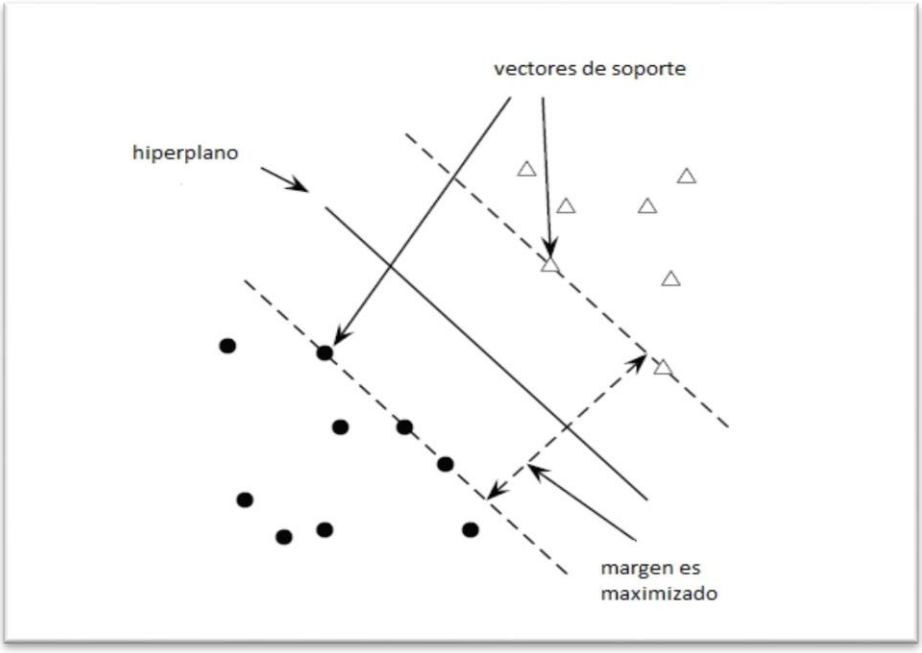

Figura 7. Maximización del margen hecho por las SVMs. Fuente: MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich.Introduction to information retrieval. Cambridge: Cambridge University Press, 2008, p 320.

La maximización del margen se puede definir como la maximización de la distancia de los dos márgenes que separan las clases (líneas punteadas en figura 7). Los cuales están dados por las ecuaciones 17. Mediante procedimientos algebraicos se puede demostrar que la distancia entre estos dos hiperplanos es la definida en la ecuación 18, obteniendo así que es necesario minimizar el valor $|w|$para maximizar el margen. Además se deben agregar restricciones para que ningún dato caiga dentro de la región definida entre ambos márgenes, por lo que es necesario agregar dos restricciones (ecuación 19) al problema.

$$w \cdot x + b = 1$$
$$w \cdot x + b = -1$$

Ecuación 17. Ecuaciones de los márgenes que separan los datos

$$M = \frac{2}{|w|}$$

Ecuación 18. Calculo de la distancia entre los dos márgenes

$$w \cdot x_i - b \geq 1 \quad \forall x_i |\, y_i = 1$$
$$w \cdot x_i - b \leq -1 \quad \forall x_i |\, y_i = -1$$

Ecuación 19. Restricciones para la separación de datos por margen

Así, el proceso de aprendizaje y clasificación de las máquinas de soporte vectorial es realizado a través de la minimización de la expresión en la ecuación 20. Esta minimización presenta dificultades a resolverse debido a que depende de la normal del vector w, la cual involucra raíz cuadrada. Por esto es necesario cambiar el $|w|$ por $\frac{1}{2}|w|$ lo cual no alteraría los resultados del problema de

optimización y lo haría más viable para su solución (ecuación 21). Este problema de minimización puede ser resuelto mediante multiplicadores de LaGrange o programación cuadrática

$$\min_{w} |w| \quad sujeto\ a:$$
$$w \cdot x_i - b \geq 1 \quad \forall x_i |\ y_i = 1$$
$$w \cdot x_i - b \leq -1 \quad \forall x_i |\ y_i = -1$$

Ecuación 20. Función de minimización usada por la SVM

$$\min_{w} \frac{1}{2}|w|^2 \quad sujeto\ a:$$
$$y_i(w \cdot x_i - b) \geq 1 \quad \forall (x_i, y_i)$$

Ecuación 21. Función de minimización simplificada usada por las SVM

Anteriormente se hizo una suposición que limita en gran medida la efectividad de las SVM: el conjunto de datos es linealmente separable. A continuación se explicara cómo resolver el problema de la no separación lineal mediante el uso de kernels.

Kernel.

Supongamos que los datos a clasificar no sean linealmente separables, como los que se muestran en la figura 9. Al no ser linealmente separables no existe ningún hiperplano capas de dividir los datos por clases. Un kernel se define como una función que computa el producto interno de los datos en un espacio de característica de dimensiones mayor (46). Con el uso de kernel, se pueden llevar

todos los datos a un espacio de característica de dimensión mayor (figura 8) de tal forma que estos sean linealmente separables y así se puedan aplicar toda la teoría de las SVM vista en el apartado anterior.

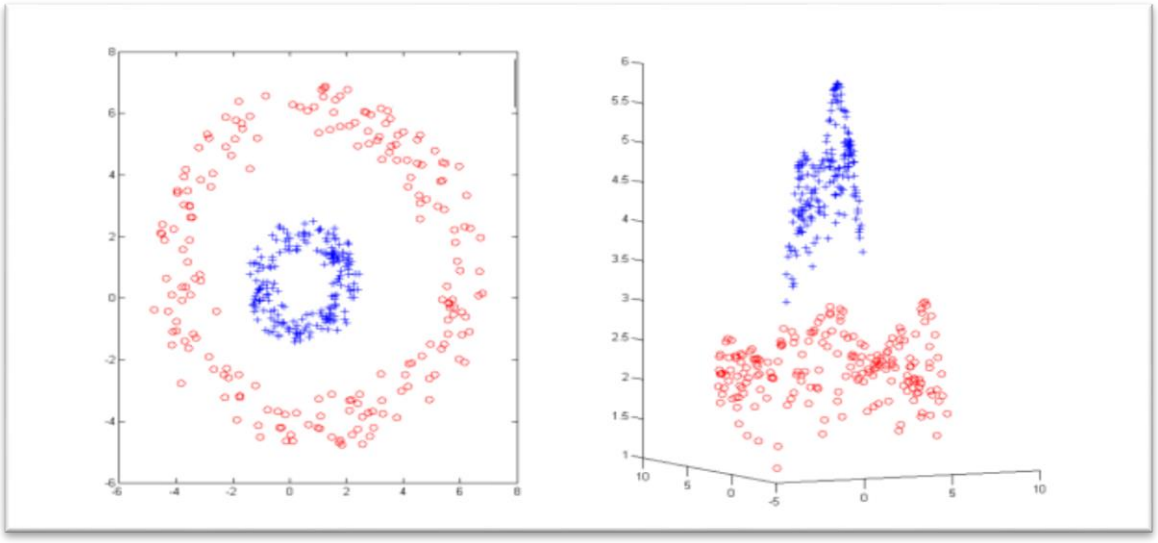

Figura 8. Transformación del espacio mediante kernels. Izquierda: datos originales. Derecha: Datos transformados. Fuente: FLETCHER, Tristan. Support vector machines explained. Tutorial paper., Mar, 2009.

Existe diversidad de funciones que se pueden utilizar como kernel para transformar los datos. Básicamente para que una función pueda ser utilizada como kernel debe satisfacer el teorema de Mercer (47) Las funciones kernel más comunes utilizadas con las máquinas de soporte vectorial se muestran en la tabla 3.

Nombre	Expresión
Kernel polinomial	$(x^T x_i + 1)^p$
Kernel gausiano	$e^{(-\gamma \lvert x - x_i \rvert^2)}$
Kernel sigmoide	$\tanh(\beta_0 x^T x_i + \beta_1)$

Tabla 3. Tipos de kernel usados con SVM

Margen suave.

Supongamos que los datos a clasificar sean linealmente separables como se muestra en la figura 6; sin embargo en el proceso de construcción del conjunto de datos se etiquetaron erróneamente unos valores a clasificar como se muestra en la figura 9, haciendo imposible separar por completo las dos clases. En estos casos en que los datos no son completamente separables, el enfoque estándar es permitir que la frontera de decisión cometa unos cuantos errores y después se hace pagar un costo por cada ejemplo mal clasificado que depende de que tan lejos esté de cumplir el requisito de la frontera de decisión. Para implementar esto se introduce una variable de holgura (48) ξ_i en la función objetivo, de tal forma que el dato de entrenamiento x_i no cumpla los requisitos del margen generando un costo para la función objetivo proporcional al valor de ξ_i. Introduciendo esta variable de holgura la función objetivo de las SVM queda formulada de la siguiente manera:

$$\min_{w,\xi_i} \frac{1}{2}|w|^2 + C\sum_i \xi_i \quad sujeto\ a:$$

$$y_i(w \cdot x_i - b) \geq 1 - \xi_i \quad \forall (x_i, y_i)$$

Ecuación 22. Función de minimización de la SVM usando margen suave

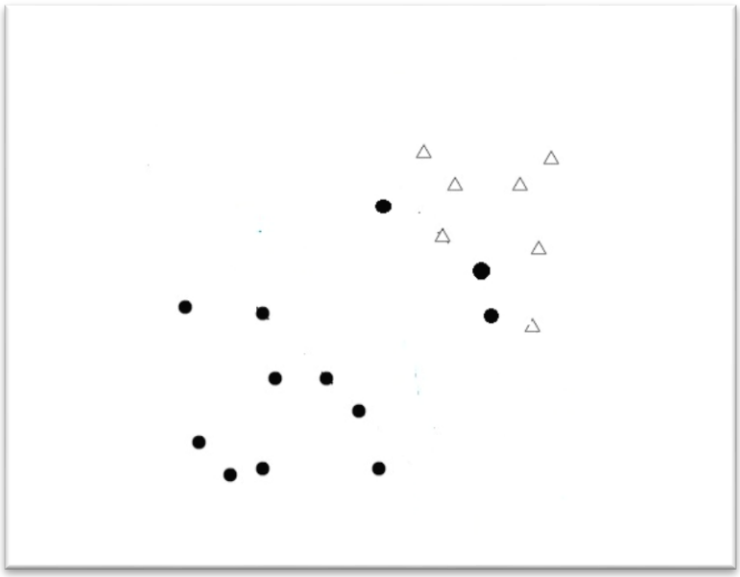

Figura 9. Conjunto de datos que no son linealmente separables. Fuente: Elaboracion propia.

El problema de optimización se convierte en una negociación entre que tan lejos se puede hacer el margen versus cuantos puntos tienen que ser movidos alrededor para permitir este margen. El parámetro C en la ecuación 22 es un término de regularización que básicamente provee una forma de evitar el sobre-entrenamiento, un problema habitual en los algoritmos de Machine Learning.

13.2 IMPLEMENTACION DE LOS MODELOS DE MACHINE LEARNING.

Para la construcción de todos los modelos de Machine Learning se utilizara. scikit-learn (49), una librería open source que tiene implementado la mayoría de estos algoritmos en el lenguaje de programación python. Todos los clasificadores de esta librería proveen métodos para entrenar los clasificadores y para predecir la

categoría de un patrón desconocido. Además se utiliza la librería científica numpy que provee un gran número de funciones que facilitan el manejo de objetos matemáticos como vectores, matrices, etc.

De la familia de clasificadores bayesianos se escogieron Multinomial Naive Bayes y Bernoulli Naive Bayes ya que estos son los más utilizados en la tarea de clasificación de texto. Mediante las clases `sklearn.naive_bayes.MultinomialNB` y `sklearn.naive_bayes.BernoulliNB` se utiliza la implementación de estos algoritmos.

Para las máquinas de soporte vectorial se utiliza una implementación de tipo C-SVM, basada en la librería lib-svm (50), mediante la clase `sklearn.svm.SVC`. Usando el constructor de la clase se especifican los parámetros del clasificador como: parámetro de regulación, tipo de kernel, parámetros del kernel, etc. Debido a que las SVMs no permiten de forma natural la clasificación entre múltiples clases, se es necesario aplicar un enfoque one-against-one (51), en cual consiste en crear para cada pareja de categorías una SVM que aprenda a separar datos entre las dos categorías. Esto quiere decir que para n clases, $\frac{n(n-1)}{2}$ SVMs son construidos y la clasificación de un patrón desconocido es hecha de acuerdo al máximo de votos, donde cada SVM vota por una clase. Afortunadamente scikit-learn implementa de forma interna este enfoque permitiendo que las SVMs trabajen con múltiples categorías.

En el caso de que los clasificadores necesiten que se le provean valores a parámetros que rigen el proceso de clasificación, el valor de estos parámetros es determinado mediante una búsqueda exhaustiva en la cual se encuentra los mejores parámetros con mejores resultados para el clasificador, esto se conoce

como selección del modelo más adecuado. Para la búsqueda de los parámetros se utiliza el algoritmo grid-search (52) con validación cruzada k-fold. scikit-learn implementa este algoritmo a través del modulo **`grid_search.GridSearchCV`** con el que se utiliza k=5 para la validación cruzada.

14. ENTRENAMIENTO Y EVALUACIÓN.

14.1 ENTRENAMIENTO

Una vez definido cada una de las etapas a realizar en la construcción de los modelos, se procede a definir el esquema en cómo estos son ejecutados entre sí. La figura 10 presenta el proceso completo para la creación de los modelos. En primera instancia se le es aplicado el pre-procesamiento al conjunto de reclamos, para luego convertirlos a una representación vectorial donde solo se tendrán en cuenta las características más relevantes determinadas por el método de selección de características. Una vez obtenido el conjunto de datos compuestos por vectores, estos son divididos en tres partes: dos para generar los modelos y la tercera para evaluar su rendimiento y tener una medida de que también funcionan.

Una vez programado este proceso, se procede a realizar la ejecución usando cuatro tipos de modelos: Multinomial Naive Bayes, Bernoulli Naive Bayes, SVM con kernel lineal y SVM con kernel gaussiano. Como primera instancia se medirá el rendimiento de los clasificadores usando los dos tipos de representación de texto: bag of words y TF-IDF. Luego se determinara el tamaño óptimo de los features a utilizar con el objetivo de maximizar la eficacia de los clasificadores. Para todo esto se utiliza el conjunto de validación para la comparación de los resultados. En la tabla 4 se muestra los resultados de los clasificadores usando los diferentes tipos de representación de texto.

A partir de los resultados obtenidos se selecciona la representación que mejor trabaja con cada clasificador. Por lo que para Multinomial Naive Bayes se utiliza

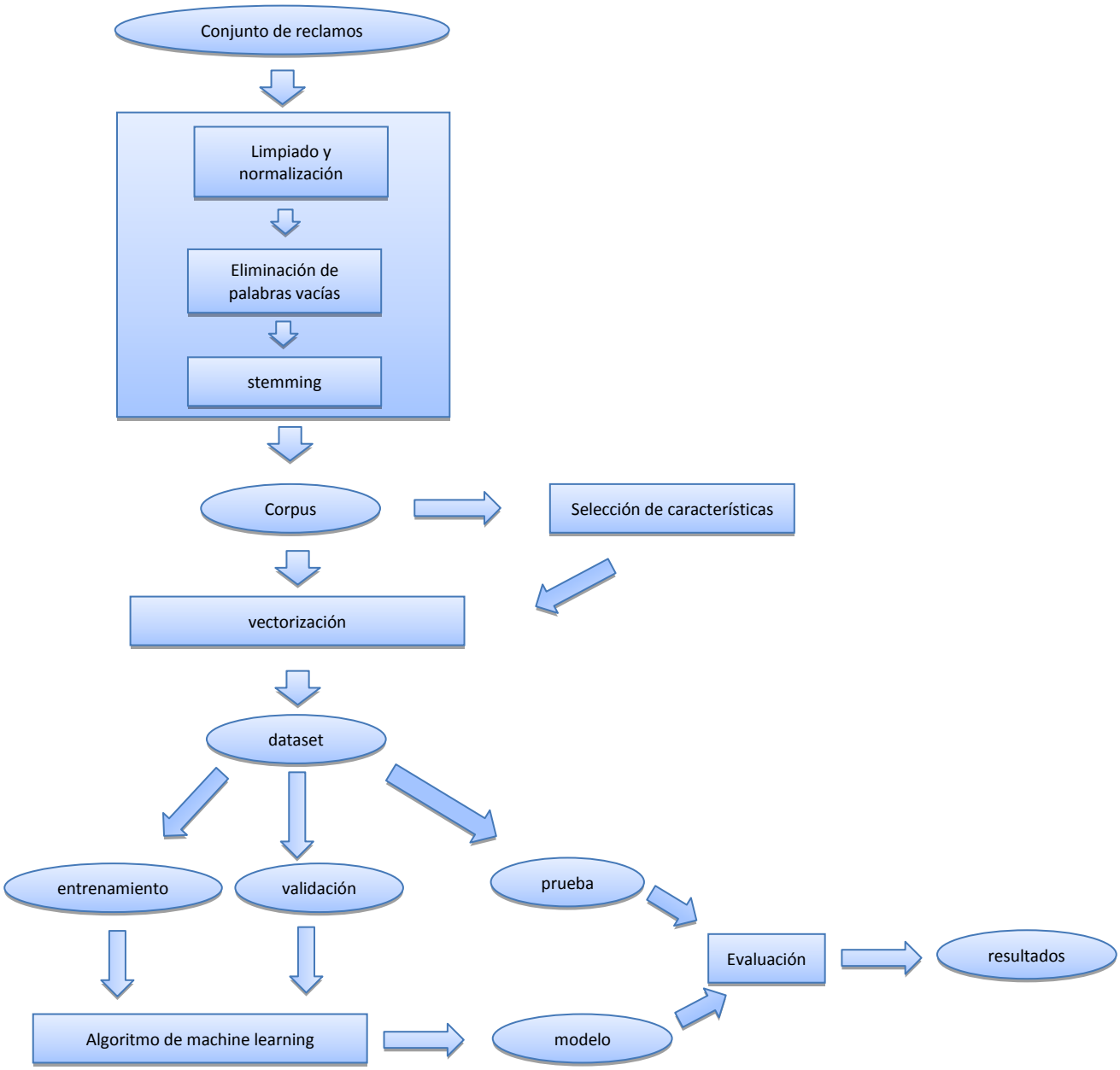

Figura 10. Diagrama de construcción de los modelos. Fuente: Elaboración propia.

Bag of Words. Para Bernulli Naive Bayes se puede utilizar Bag of Words o TF-IDF. Y para las SVM con kernel lineal y gaussiano se utiliza TF-IDF.

	precision/recall/F1		
	Bag of Words	**Bag of Words normalizado**	**TF-IDF**
Multinomial Naive Bayes	0.73/0.74/0.73		0.62/0.67/0.63
Bernoulli Naive Bayes	0.74/0.72/0.70		0.74/0.72/0.70
SVM-lineal	0.41/0.52/0.45	0.57/0.52/0.48	0.62/0.65/0.62
SVM-gaussiano	0.40/0.50/0.44	0.34/0.35/0.33	0.61/0.65/0.61

Tabla 4. Resultados de los clasificadores con diferente representación de datos evaluados sobre el conjunto de validación.

Teniendo una representación de texto adecuada para cada clasificador, se procede a determinar el valor óptimo de features a extraer con nuestro algoritmo de feature selection. Para esto se registran los resultados obtenidos con diferentes tamaños de features. Como nuestro algoritmo de selección de features involucra una aleatoriedad en su proceso, para cada número de features se corre el algoritmo 5 veces y se promedia las medidas F1 obtenidas. Los resultados obtenidos se muestran en la figura 11. En la figura 12 se muestra el promedio de los 4 clasificadores de las desviaciones estándar de la medida F1 entre las 5 ejecuciones del algoritmo de feature selection para diferentes números de features. Como se puede observar la mayor desviación es de 0.03 (3%), por lo tanto se puede inferir que a pesar de que nuestro algoritmo de feature selection introduce una aleatoriedad en el proceso de selección, los resultados obtenidos son consistentes.

De la figura 11, se puede apreciar que Multinomial Naive Bayes es el algoritmo que menos sufre la variación en el número de features ya que sus resultados no

son tan variados. Los mejores resultados son obtenidos cuando el número de feature es 800 y la medida F1 es de 0.72, resultado bastante cercano al obtenido con el mismo algoritmo sin utilizar selección de features. Por otro lado Bernoulli Naive Bayes presenta mayor variación con respecto al número de features a utilizar; Su mejor resultado es obtenido cuando el número de feature es 1800 y no es tan difenrente al obtenido sin usar selección de features. Para las máquinas de soporte vectorial, se observa que estas presentan resultados muy similares usando kernels líneal y gaussiano; Además obtienen sus mejores resultados con una medida F1 de 0.69 y 0.68 usando kernel lineal y gaussiano respectivamente. Para las SVM es una gran mejora en su eficacia comparándolos con los resultados en donde se usan todos los features, tabla 4.

Por último, usando el número de features optimo y la representación de texto óptima para cada clasificador, se muestran los mejores parámetros de cada clasificador obtenidos mediante validación cruzada k-fold con k=5. En la tabla 5 se muestran los parámetros para cada clasificador.

Clasificador	Representación	features	Parámetros obtenidos
Multinomial Naive Bayes	Bag of words	800	$\alpha = 0.2$
Bernoulli Naive Bayes	Bag of words	1800	$\alpha = 0.1$
SVM-lineal	TF-IDF	600	$C = 2.0$
SVM-gaussiano	TF-IDF	700	$C = 2.0,\quad \gamma = 0.5$

Tabla 5. Parámetros para los clasificadores obtenidos usando validación cruzada sobre el conjunto de entrenamiento

Figura 11. Medida F1 de los clasificadores con diferente número de features. Fuente: Elaboración propia.

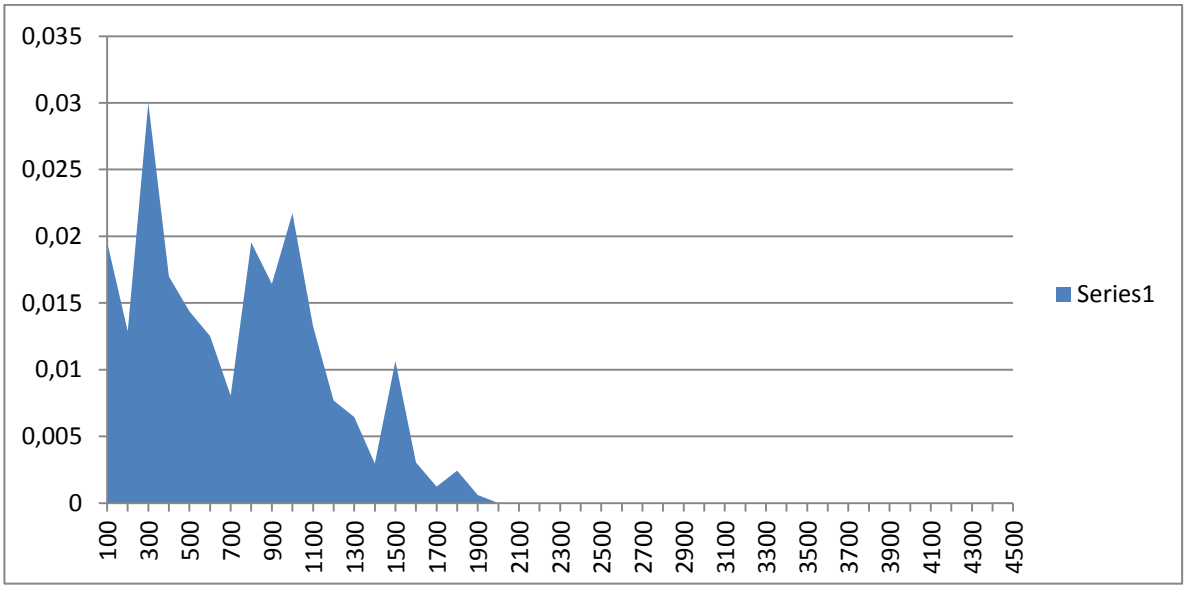

Figura 12. Media de las desviaciones estándar de la medida F1 entre los 4 clasificadores para diferentes números de features. Fuente: Elaboración propia.

14.2 EVALUACIÓN

A partir de los parámetros de la tabla 5, se evaluaran los clasificadores sobre el conjunto de pruebas para tener una medida real de que también funcionan.

	Precision	Recall	F1	Número de datos
Mantenimiento de planta física	0.68	0.72	0.70	29
Servicios bibliotecarios	0.77	0.96	0.85	48
Servicios informáticos	0.33	0.15	0.21	13
Medición análisis y mejora	1.00	0.82	0.90	17
Bienestar universitario	0.50	0.50	0.50	10
Servicios Administración financiera	0.71	0.83	0.77	18
Servicios Académicos	1.00	0.22	0.36	9
Promedio/total	0.73	0.73	0.70	144

Tabla 6. Resultados finales usando Multinomial Naive Bayes

	Precision	Recall	F1	Número de datos
Mantenimiento de planta física	0.65	0.83	0.73	29
Servicios bibliotecarios	0.80	0.85	0.83	48
Servicios informáticos	0.29	0.15	0.20	13
Medición análisis y mejora	0.88	0.82	0.85	17
Bienestar universitario	0.57	0.40	0.47	10
Servicios Administración financiera	0.61	0.78	0.68	18
Servicios Académicos	1.00	0.33	0.50	9
Promedio/total	0.71	0.71	0.69	144

Tabla 7. Resultados finales Usando Bernoulli Naive Bayes

	Precision	Recall	F1	Número de datos
Mantenimiento de planta física	0.59	0.79	0.68	29
Servicios bibliotecarios	0.81	0.90	0.85	48
Servicios informáticos	0.50	0.23	0.32	13
Medición análisis y mejora	0.94	0.94	0.94	17
Bienestar universitario	0.22	0.20	0.21	10
Servicios Administración financiera	0.72	0.72	0.72	18
Servicios Académicos	0.50	0.11	0.18	9
Promedio/total	0.68	0.70	0.68	144

Tabla 8. Resultados finales usando SVM con kernel lineal

Tabla 9. Resultados finales usando kernel gaussiano

	Precision	Recall	F1	Número de datos
Mantenimiento de planta física	0.58	0.86	0.69	29
Servicios bibliotecarios	0.81	0.96	0.88	48
Servicios informáticos	0.67	0.15	0.25	13
Medición análisis y mejora	0.88	0.88	0.88	17
Bienestar universitario	0..25	0.10	0.14	10
Servicios Administración financiera	0.78	0.78	0.78	18
Servicios Académicos	0.50	0.11	0.18	9
Promedio/total	0.70	0.72	0.68	144

14.3 ANÁLISIS DE RESULTADOS.

A partir de los resultados obtenidos se puede ver que la eficacia de los 4 clasificadores produce resultados muy similares entre ellos. Sin embargo Los algoritmos Naive Bayes son los que mejores resultados generan en el conjunto de prueba a pesar de su simplicidad en el proceso de clasificación. Esto concuerda con las hipótesis propuestas por Ng (53) la cual establece que Naive bayes es un algoritmo de “alto-bias” lo cual lo hace capaz de obtener buenos resultados con pocos datos de entrenamiento incluso cuando la dimensión de los datos es bastante alta.

Las máquinas de soporte vectorial también son capaces de obtener resultados similares a los de Naive Bayes pero esto es en parte por el algoritmo de selección de features que se usó para reducir la dimensionalidad de los datos. Como se muestra en la tabla 4, los resultados de las SVMs no son muy prometedores cuando se usa los 4622 features de los datos. Esto se explica con el hecho de que las SVMs son propensas al sobre-entrenamiento cuando el conjunto de datos es muy pequeño y la dimensión de los datos es muy grande como es el caso de los problemas de clasificación de texto. Por otra parte la variable kernel no jugó un papel decisivo en los resultados ya que el uso de un kernel lineal (no kernel) y uno gaussiano no hizo ninguna diferencia en la eficacia de las SVMs. La alta dimensionalidad del problema hace innecesario el uso de un kernel.

En la figura 13 se ilustran los resultados de los clasificadores evaluados en cada proceso. Se puede observar que los peores resultados son obtenidos para los procesos “Servicios informáticos”, “Bienestar universitario” y “Servicios académicos” los cuales son las clases con menor cantidad de datos en el conjunto de datos; sin embargo en los dos últimos los algoritmos de Naive Bayes logran obtener una medida F1 de 0.5. Algo interesante a destacar es que el proceso “Medición análisis y mejoras” con 57 datos obtiene mejores resultados que el proceso “Servicios bibliotecarios” el cual posee 161. Esto indica que no solo es necesario tener gran cantidades de datos para obtener buenos resultados si no que es necesario que los datos a utilizar sean los suficientemente representativos como es el caso de la categoría “Medición análisis y mejoras”.

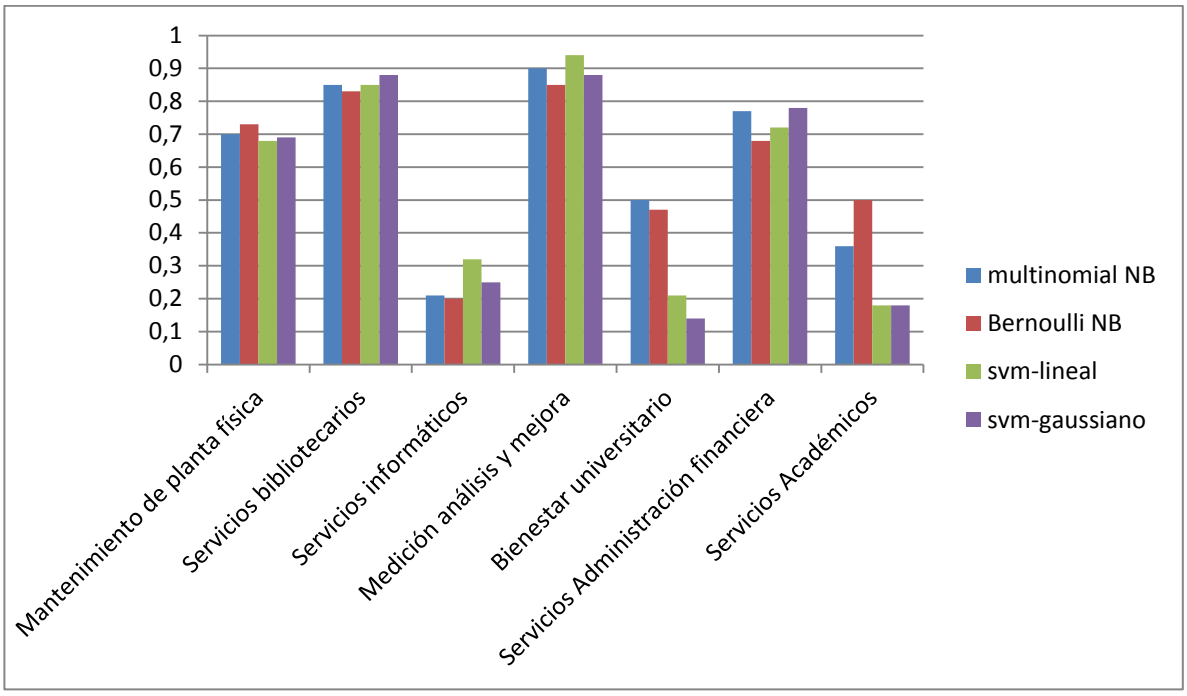

Figura 13. Resultados de los clasificadores evaluados en cada proceso. Fuente: Elaboración propia.

Algo a aclarar es que dado que Multinomial Naive Bayes es el que mejores resultados genera, al recomendar este modelo para futuras tareas de clasificación de reclamos los resultados que se obtendrán serán un poco menores a los obtenidos en nuestras pruebas. Esto se debe a que al usar todos los clasificadores sobre el conjunto de pruebas y recomendar el que mejor rendimiento tiene, se ha sesgado los resultados porque se ha escogido a posteriori el clasificador al que mejor le fue sobre unos datos específicos, o en otras palabras el que mejor se ajusta a ese conjunto especifico.

15. CONCLUSIONES

La categorización automática de texto es ahora un mayor campo de investigación dentro de los sistemas de información debido a que sus campos de aplicaciones son numerosos e importantes, y dada la proliferación de documentos en formato digital, están obligados a aumentar tanto en número como en importancia. La mayoría de investigaciones realizadas en torno al procesamiento del lenguaje natural han sido para el idioma inglés obteniendo grandes avances en los diferentes niveles lingüísticos. Estos avances combinados con técnicas de Machine Learning han logrado generar modelos altamente precisos que pueden competir en la sociedad contemporánea. Sin embargo para el lenguaje español, el procesamiento de texto es más difícil que el inglés debido a la complejidad de la estructura lingüística del lenguaje. En este proyecto de investigación se presentó varios modelos de clasificación para predecir el proceso al cual pertenece un reclamo perteneciente a las instalaciones de la Universidad Tecnológica de Bolívar. A pesar de las limitaciones de las herramientas para el procesamiento del lenguaje español, se pudo obtener mediante técnicas de Machine Learning resultados bastante prometedores que fueron de una u otra forma limitados por la cantidad de datos que se pudo recolectar para la investigación.

Con base en las experiencias obtenidas a lo largo de la presente investigación, se pudo recolectar varias conclusiones expuestas a continuación.

- A pesar de las pocas herramientas para el procesamiento del lenguaje español existentes, las técnicas de Machine Learning puede proveer excelente resultados para los problemas de clasificación de textos para este lenguaje.
- No es necesario utilizar procesamientos y representaciones complejas de textos para obtener buenos resultados para clasificar texto en español. El

uso de técnicas simples de limpieza y normalización de texto y una simple representación de texto, es suficiente para obtener buenos resultados mediante algoritmos de Machine Learning.

- Basados en los resultados de los clasificadores medidos por procesos, se puede establecer que para tener buenos resultados clasificando reclamos de un proceso, se es necesario tener como mínimo alrededor de 60 reclamos pertenecientes a dicho proceso.
- Es necesario recolectar mayor cantidad de reclamos ya que algunos procesos poseen muy pocos. Al poseer mayor cantidad de datos se puede construir clasificadores capaces de trabajar con los 43 procesos existentes en la Universidad Tecnológica de Bolívar.

REFERENCIAS

(1) Calidad online: sugerencias y reclamos [en línea] [citado el 31 de octubre, 2012] Disponible en: <url: http://calidadonline.unitecnologica.edu.co:8080/CalidadOnline/sugerenciareclamo >

(2) HERNANDEZ, Sampier y FERNANDEZ, Carlos. Metodología de la investigación. Cuarta Edición. Mexico D.F.: Mc Graw Hill, 2006.p 103. ISBN 978-970-10-5753-7.

(3) Real academia de la lengua española [En línea]. [citado el 08 de julio, 2012] Disponible en: <url: http://lema.rae.es/drae/?val=modelo >

(4) Real academia de la lengua española [En línea]. [citado el 08 de julio, 2012] Disponible en: <url: http://lema.rae.es/drae/?val=aplicacion >

(5) Real academia de la lengua española [En línea]. [citado el 08 de julio, 2012] Disponible en: <url: http://lema.rae.es/drae/?val=generalizar >

(6) MITCHELL, T., et al. Machine Learning [Aprendizaje automático]. En: Annual review of computer science. 1990. Vol. 4, p. 417-433.

(7) KOTSIANTIS, S.; KANELLOPOULOS, D. y PINTELAS P. Data preprocessing for supervised learning [Procesamiento de datos para aprendizaje supervisado]. En: International Journal of Computer Science. 2006; Vol. 1, p. 111–117.

(8) ALPAYDIN, Ethen. Introduction. En: Introduction to machine learning. 2 ed. Massachusetts: the MIT Press, 2010. P. 5-6.

(9) BISHOP, Christopher M. Introduction. En: Pattern Recognition and Machine Learning. Singapore: Springer, 2006. P. 6-7.

(10) HAYKIN, S. En: Neural networks and learning machines [Redes neuronales y aprendizaje automático]. 3 ed. New York, USA: Prentice Hall, 2009. P 14-16.

(11) BECKER, S. Unsupervised learning procedures for neural networks [aprendizaje no supervisado par redes neuronales]. En: International journal of Neural systems. Octubre, 1991. Vol. 2, p. 17-31

(12) ZHU, Xiaojin. Semi-Supervised Learning Literature Survey [aprendizaje semi-supervisado]. En: Computer Sciences Technical Report. Diciembre, 2006. Vol. 1530, p. 4-10.

(13) LOUNIS, H. y BISSON, G. Evaluation of learning system: an artificial data-based approach [evaluación de sistemas de aprendizaje: un enfoque basado en datos artificiales]. En: Lecture Notes in Computer Science. Abril, 1991. Vol. 482, p. 466-467.

(14) LIDDY, E. D. Natural language processing [procesamiento del lenguaje natural]. En: Encyclopedia of Library and Information Science. 1998. 2 ed. New York: Marcel Decker, 2005.

(15) CHOWDHURY, G. Natural language processing [procesamiento del lenguaje natural]. En: Annual Review of Information Science and Technology. 2003. Vol. 37, p. 51-89.

(16) MOURÃO, Fernando, et al. Understanding Temporal Aspects in Document Classification [entendiendo aspectos temporales en la clasificación de documentos]. En: WSDM '08 Proceedings of the international conference on Web search and web data mining. 2008. P. 159-170.

(17) BORKO, H. y BERNICK, M. Automatic document classification [clasificación automática de documentos]. En: J. ACM. 1963. Vol. 10, p. 151–162.

(18) Herramientas para sistemas de gestión de calidad iso-9001 [en línea]. [citado el 05 de agosto, 2012] Disponible en: <url: http://www.normas9000.com/que-es-iso-9000.html >

(19) Norma ISO 9001:2008 [en línea]. [citado el 05 de agosto, 2012] Disponible en: <url: http://www.esu.com.co/esu/documentos/normatividad/Norma%20ISO9001%202008.pdf>

(20) LARKEY, L. S. 1999. A patent search and classification system[una busqueda de patentes y sistemas de clasificación]. En: Proceedings of DL-99, 4th ACM Conference on Digital Libraries. 1999. p. 179–187.

(21) VAINE, Stijn, et al. A comparasion of State-of-the-art classification techniques for expert automobile insurance claim fraud detection[una comparación del estado del arte de técnicas de clasificación para la detección de fraudes en reclamos de seguros automovilísticos]. En: The Journal of Risk and Insurance. 2002. Vol. 69, No 342, p. 373-1.

(22) KWON, N., et al. Identifying and classifying subjective claims [identificando y clasificando reclamos subjetivos]. En: Proceedings of the 8th Annual International Digital Government Research Conference. 2006. P. 76–81.

(23) YU, Bei; KAUFMANN, Stefan y DIERMEIER, Daniel. Classifying party affiliation from political speech[Clasificación de afiliación a un partido a partir de un discurso político]. En: Journal of Information Technology and Politics. 2008. Vol. 5(1), p. 33–48.

(24) NGAI, E.W.T., et al. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature[La aplicación de técnicas de minería de datos en la detección de fraude financiero: un

framework de clasificación y una revisión academica de la literatura]. En: Journal decision Support System. Febrero, 2011. Vol. 50, p. 559-569.

(25) HOCH, R. Using IR Techniques for Text Classification in Document Analysis[Usando técnicas de recuperación de la información para la clasificación de texto en análisis de documentos]. En: Proc. 17th Ann. Int'l ACM SIGIR Conf.Research and Development in Information Retrieval. Julio, 1994. p. 31-40.

(26) PEREA, J.M., et al. Categorización de textos biomédicos usando UMLS. En: Procesamiento del Lenguaje Natural. 2008. Vol. 40, p. 121-127.

(27) MAÑA, M., et al. Los proyectos SINAMED e ISIS: Mejoras en el Acceso a la Información Biomédica Mediante la Integración de Generación de Resúmenes, Categorización Automática de Textos y Ontologías. En: Actas del XXII Congreso de la Sociedad Española de Procesamiento del Lenguaje (SEPLN). septiembre 2006. Vol. 37, p. 353-357.

(28) CLACK, C., et al. Autonomous document classification for business[Clasificación automática de documentos para empresas]. En: Proceedings of the 1st International Conference on Autonomous Agents. 1997. p. 201–208.

(29) YOLIS, E., et al. Algoritmos Genéticos aplicados a la Categorización Automática de Documentos. En: Revista Eletrônica de Sistemas de Informação. 2003. Vol. 2, p. 49-63.

(30) HAYES, P. J., et al. Tcs: a shell for content-based text categorization[Clasificación de texto: a shell para categorización de texto basado en contenido]. En: Proceedings of CAIA-90, 6th IEEE Conference on Artificial Intelligence Applications. Mayo, 1990. p. 320–326.

(31) LANA, S.; COLLADA, J. Villena y GONZALES, J. Generación automática de reglas de categorización de texto en un método híbrido basado en aprendizaje. En: Procesamiento del Lenguaje Natural. Septiembre, 2011. Vol. 47, p. 231-237

(32) Snowball [En línea]. [Citado el 9 de febrero de 2013] Disponible en: <url: http://snowball.tartarus.org/algorithms/spanish/stemmer.html>

(33) CHUM, Ondrej; PHILBIN, James; ZISSERMAN, Andrew. Near duplicate image detection: min-hash and tf-idf weighting. En *Proceedings of the British Machine Vision Conference*. 2008. p. 4.

(34) YU, Lei; LIU, Huan. Feature selection for high-dimensional data: A fast correlation-based filter solution. En *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*. 2003. p. 856.

(35) FORMAN, George. A pitfall and solution in multi-class feature selection for text classification. En *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004. p. 38.

(36) FORMAN, George. Choose your words carefully: An empirical study of feature selection metrics for text classification. En *Principles of Data Mining and Knowledge Discovery*. Springer Berlin Heidelberg, 2002. p. 150-162.

(37) FORMAN, George. BNS feature scaling: an improved representation over tf-idf for svm text classification. En *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008. p. 263-270.

(38) Un algoritmo para computar la distribución normal estándar de la función inversa de probabilidad acumulativa [En línea]. [citado el 22 de enero, 2013] Disponible en: http://home.online.no/~pjacklam/notes/invnorm/index.html

(39) DAVIS, Jesse; GOADRICH, Mark. The relationship between Precision-Recall and ROC curves. En *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006. p. 233-240.

(40) CHEN, Tsong Yueh; KUO, Fei-Ching; MERKEL, Robert. On the statistical properties of the f-measure. En *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*. IEEE, 2004. p. 146-153.

(41) BISHOP, Christopher M. Introduction. En: Pattern Recognition and Machine Learning. Singapore: Springer, 2006. P. 21-24.

(42) Russell, Stuart; Peter Norvig. En: Artificial Intelligence: A Modern Approach. Prentice Hall. 2002. P. 478.

(43) MCCALLUM, Andrew, et al. A comparison of event models for naive bayes text classification. En *AAAI-98 workshop on learning for text categorization*. 1998. p. 41-48.

(44) FIELD, A David. En: Laplacian smoothing and Delaunay triangulations. En: Internacional Journal for Numerical Methods in Biomedical Engineering. 2005. Vol 4, p 709-712.

(45) BOSER, Bernhard E.; GUYON, Isabelle M.; VAPNIK, Vladimir N. A training algorithm for optimal margin classifiers. En *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992. p. 144-152.

(46) HAYKIN, S. En: Neural networks and learning machines [Redes neuronales y aprendizaje automático]. 3 ed. New York, USA: Prentice Hall, 2009. P 282.

(47) HAYKIN, S. En: Neural networks and learning machines [Redes neuronales y aprendizaje automático]. 3 ed. New York, USA: Prentice Hall, 2009. P 283.

(48) Hamdy A. Taha. En: Investigación de operaciones. 7 ed. Mexico: Prentice Hall, 2004. P 94.

(49) PEDREGOSA, Fabian, et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 2011, vol. 12, p. 2825-2830.

(50) CHANG, Chih-Chung; LIN, Chih-Jen. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011, vol. 2, no 3, p. 27.

(51) DEBNATH, R.; TAKAHIDE, N.; TAKAHASHI, H. A decision based one-against-one method for multi-class support vector machine. *Pattern Analysis & Applications*, 2004, vol. 7, no 2, p. 164-175.

(52) Singiresu S. Rao, S. S. Rao. En: Engineering Optimization: Theory and Practice. 4 ed. New York, USA: Willey, 2009. P314.

(53) JORDAN, A. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2002, vol. 14, p. 841.

ANEXOS

Listado 1. Reclamo

```
'''
Created on 15/12/2012

@author: Jorge
'''

class Claim:

	vocabulary = []
	count = 0

	def __init__(self, description, category):

		self.description = description
		self.category = category
		self.tag= None
		self.update_vocabulary()
		self.id = Claim.count
		Claim.count+=1

	def get_description(self):
		return self.description

	def get_category(self):
		return self.category

	def get_id(self):
		return self.id

	def set_description(self, description):
		self.description = description
		self.update_vocabulary()

	def set_category(self, category):
		self.category = category

	def set_meta_data(self, data):
		self.tag = data

	def get_meta_data(self):
		return self.tag
```

```
def save(self, output):
            output.write(self.category+'\t'+self.description+"\n")

def update_vocabulary(self):
      for token in self.description.split():
                  if token not in Claim.vocabulary:
                        Claim.vocabulary.append(token)


def vectorized_form(self):
      tokens = self.description.split()
      vector = range(len(Claim.vocabulary))
      for i in range(len(Claim.vocabulary)):
            if Claim.vocabulary[i] in tokens:
                  vector[i] = tokens.count(Claim.vocabulary[i])
            else:
                  vector[i] = 0

      return vector

def sparse_data(self):
      sparse = {}
      sparse[-1]=0.0
      tokens = self.description.split()
      for i in range(len(Claim.vocabulary)):
            if Claim.vocabulary[i] in tokens:
                  sparse[i]= float (tokens.count(Claim.vocabulary[i]))
      return sparse


def __eq__(self, other):
      return self.id == other.id

def __hash__(self):
      return hash(self.id)
```

Listado 2. Reclamo en forma vectorizada.

```
'''
Created on 15/12/2012

@author: Jorge
'''

class Example(object):

    categories = []

    def __init__(self,X, y):

        self.X = X

        if type(y)==str:
            self.y = y.replace(' ','_')
        else:
            self.y = y
        self.update_categories()

    def get_X(self):
        return self.X

    def get_y(self):
        return self.y

    def save(self, output):
        output.write(self.y+'\t'+self.description+"\n")

    def update_categories(self):
        if self.y not in Example.categories:
            Example.categories.append(self.y)

    def get_numerical_category(self):
        return Example.categories.index(self.y)

    @staticmethod
    def get_number_label(label):
        return Example.categories.index(label)

    def get_vector_X(self):
        """v=[]
        indexes = self.X.keys()
        indexes.sort()
        for i in indexes:
            v.append(self.X[i])
        return v"""
        return self.X
```

Listado 3. Clase que encapsula acceso a los datos.

```
#!/usr/bin/env python
import random
from xlrd import open_workbook
from corpus_cleaner import CorpusCleaner
from Claim import Claim
from VectorizedClaim import Example

class DatasetAccess:

	CATEGORY_COLUMN = 5
	DESCRIPTION_COlUMN = 10
	DATASET_PATH = ""
	TRAINING_SET_FILE = "../data/training_set.dat"
	VALIDATION_SET_FILE = "../data/validation_set.dat"
	TEST_SET_FILE = "../data/test_set.dat"
	TRAING_SET_PERCENT = 0.6
	VALIDATION_SET_PERCENT = 0.1
	TEST_SET_PERCENT = 0.3

	def __init__(self):
		self.training_set_path = DatasetAccess.TRAINING_SET_FILE
		self.validation_set_path = DatasetAccess.VALIDATION_SET_FILE
		self.test_set_path = DatasetAccess.TEST_SET_FILE

	def create_new_datasets(self):
		book = open_workbook('../data/Consolidado_de_quejas_2009-2012.xls')
		sheet = book.sheet_by_index(0)
		descriptions =
sheet.col_values(DatasetAccess.DESCRIPTION_COlUMN)[2:]
		categories = sheet.col_values(DatasetAccess.CATEGORY_COLUMN)[2:]
		claims = {}

		cleaner = CorpusCleaner()
		for i in range(len(descriptions)):
			desc = str(descriptions[i].encode('utf-8'))
			c = str(categories[i].encode('utf-8'))
			desc = cleaner.clear_claim(desc)
			c = cleaner.clear_claim(c, False)
			c = c.replace(' ','_')
			print c
			try:
				claims[c].append( Claim(desc, c) )
			except KeyError:
				claims[c]=[ Claim(desc, c) ]

		#mean = 18
		keys = claims.keys()
```

```
        for category in keys:
                        l = len(claims[category])
                        if l<=30:
                                del claims[category]
                                continue
                        elif l>30 and l<=50:
                                new_category= "minoriry"
                        else:
                                new_category= "mayority"

                        for e in claims[category]:
                                e.set_meta_data(new_category)

                training_set = []
                validation_set = []
                test_set = []
                for category in claims.keys():
                        docs = claims[category]
                        random.shuffle(docs)

                        i = 0
                        j =int(
        len(docs)*DatasetAccess.TRAING_SET_PERCENT )
                        training_set.extend(docs[i:j])

                        i, j = j, j + int(
        len(docs)*DatasetAccess.VALIDATION_SET_PERCENT )
                        validation_set.extend(docs[i:j])

                        i, j = j, j + int(
        len(docs)*DatasetAccess.TEST_SET_PERCENT )
                        test_set.extend(docs[i:j])

                random.shuffle(training_set)
                random.shuffle(validation_set)
                random.shuffle(test_set)

                v_training_set, v_validation_set, v_test_set =
        self.vectorizer_claims(training_set, validation_set, test_set)

                self.save_dataset(self.training_set_path,
        v_training_set)
                self.save_dataset(self.validation_set_path,
        v_validation_set)
                self.save_dataset(self.test_set_path, v_test_set)

    def save_dataset(self, file_name, dataset):
        raise NotImplementedError( "save_dataset method not implemented" )

    def get_training_set(self):
        return self.read_dataset(self.training_set_path)

    def get_validation_set(self):
        return self.read_dataset(self.validation_set_path)
```

```
	def get_test_set(self):
			return self.read_dataset(self.test_set_path)

	def get_dataset(self):
		dataset = []
		dataset.extend( self.get_training_set() )
		dataset.extend( self.get_validation_set() )
		dataset.extend( self.get_test_set() )
		return dataset

	def read_dataset(self, file_name):
		raise NotImplementedError( "read_dataset method not implemented" )

	def vectorizer_claims(self, training_set, validation_set, test_set):
		X=[]
		Y=[]
		l_training = len(training_set)
		l_validation = len(validation_set)
		l_test = len(test_set)
		for example in training_set:
			X.append(example.vectorized_form())
			Y.append(example.get_category())
		for example in validation_set:
			X.append(example.vectorized_form())
			Y.append(example.get_category())
		for example in test_set:
			X.append(example.vectorized_form())
			Y.append(example.get_category())

		m = len(X)
		examples =[]
		for i in range(m):
			e = Example(X[i], Y[i])
			examples.append(e)

		return examples[0:l_training],
examples[l_training:l_training+l_validation], examples[l_training+l_validation:
l_training+l_validation+l_test]
```

Listado 4. Clase encargada del proceso de limpiado del corpus.

```
from utilities import Utilities as util
import urllib2, urllib
import unicodedata
import re

class CorpusCleaner:

	def __init__(self):
		stop_words = util.read_document('stop_words.txt')
		stop_words = self.__remove_tilde(stop_words)
		self.stop_words_list = set(stop_words.split())

		self.pattern = re.compile('[\w]+')

	def clear_claim(self, text, stemming=True):
		text = text.lower()
		text = self.__remove_tilde(text)
		text = self.__remove_non_words(text)
		text = self.__remove_stop_words(text)
		if stemming:
			text = self.__stemming(text)
		text = self.__remove_non_words(text)
		return text

	def __remove_tilde(self, text):
		temp = ''.join((c for c in unicodedata.normalize('NFD',
unicode(text,'utf-8')) if unicodedata.category(c) != 'Mn'))
		return temp.encode('utf-8')

	def __remove_non_words(self, text):
		tokens = self.pattern.findall(text)
		text = ''
		for t in tokens:
			text+= t+' '
		text.encode('utf-8')
		return text


	def __stemming(self, text):
		tokens = text.split()
		words = ""
		for t in tokens: words = words+t+'_'
		try:
			params = urllib.urlencode({"words": words.encode('utf-8')})
			f = urllib2.urlopen("http://127.0.0.1/stemming.php?"+params)
			salida = f.read()
			f.close()
			salida = salida.replace(',',' ')
			return salida
```

```
except urllib2.HTTPError, e:
                    print "Ocurrio un error"
                    print e.code
             except urllib2.URLError, e:
                    print "Ocurrio un error"
                    print e.reason

             return text

def __remove_stop_words(self, text):
      text_list = text.split()
      for word in self.stop_words_list:
             if word in text_list:
                    text_list.remove(word)
      new_text=''
      for word in text_list:
             new_text+=word+' '
      new_text.encode('utf-8')
      return new_text
```

Listado 5. Diferentes implementaciones para el formato del dataset.

```
'''
Created on 16/12/2012

@author: Jorge
'''
from create_data_sets import DatasetAccess
from VectorizedClaim import Example
from dataset import convert_sparse_to_vector

class CustomDataSet(DatasetAccess):

    def __init__(self):
        self.training_set_path = '../data/customdata/training_set.tab'
        self.validation_set_path = '../data/customdata/validation_set.tab'
        self.test_set_path = '../data/customdata/test_set.tab'

    def save_dataset(self, file_name, dataset):
        file_object = open(file_name, "w")
        for example in dataset:
            file_object.write( str(example.get_y()))
            file_object.write(' ')
            x = example.get_vector_X()
            for feature in x:
                file_object.write(str(feature)+' ')
            file_object.write('\n')
        file_object.close()

    def read_dataset(self, file_name):
        dataset=[]
        f = open(file_name)
        for line in f:
            tokens= line.split()
            y = tokens[0]
            x = [float(t)  for t in tokens[1:]]
            dataset.append(Example(x, y))
        return dataset


class DatasetForOrange(DatasetAccess):

    def __init__(self):
        self.training_set_path = '../data/orange/training_set.tab'
        self.validation_set_path = '../data/orange/validation_set.tab'
        self.test_set_path = '../data/orange/test_set.tab'

    def save_dataset(self, file_name, dataset):
        output = open(file_name,'w')
        sparse = [e.sparse_data() for e in dataset]
        vector_list = convert_sparse_to_vector(sparse)
        n = len(vector_list[0])
```

```
        header = 'category'
                for i in xrange(n):
                    header+='\tw'+str(i)
                header+='\n'
                output.write(header)
                output.write('d\t')
                output.write(''.join( 'c\t' for i in xrange(n) ))
                output.write('\n')
                output.write('class\n')

                for i in range(len(dataset)):
                    line = dataset[i].get_y().encode('utf-8').replace(' ','_')
                    for count in vector_list[i]:
                        line+='\t'+str(count)
                    line+='\n'
                    output.write(line)

                output.close()



class DatasetForWeka(DatasetAccess):

    def __init__(self):
        self.training_set_path = '../data/weka/training_set.arff'
        self.validation_set_path = '../data/weka/validation_skjet.arff'
        self.test_set_path = '../data/weka/test_set.arff'

    def save_dataset(self, file_name, dataset):
        output = open(file_name,'w')
        output.write('@RELATION claims\n\n')
        sparse = [e.sparse_data() for e in dataset]
        vector_list = convert_sparse_to_vector(sparse)
        n = len(vector_list[0])
        for i in xrange(n):
            output.write(' @ATTRIBUTE w'+str(i)+'  NUMERIC\n')
        temp = '@ATTRIBUTE class {'
        for c in Example.categories:
            temp+= c+','
        temp+= '}\n\n'
        output.write(temp)
        output.write('@DATA\n')

        for i in range(len(dataset)):
            line =''
            for count in vector_list[i]:
                line+=str(count)+','
            line+= dataset[i].get_y().encode('utf-8')+'\n'
            output.write(line)

        output.close()

if __name__ == '__main__':
    data = CustomDataSet()
    data.create_new_datasets()
```

Listado 6. Ranking de feature mediante BNS para dos categorías.

```
'''
Created on 5/01/2013

@author: Jorge
'''
import math

class BinormalSeparation(object):
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
        pass

    def get_index_features(self, X, Y):
        indexes = []
        features = self.get_feature_score(X, Y)
        for f in features:
            if f.score>=0.5:
                indexes.append(f.index)
        return indexes

    def get_feature_score(self,X,Y):
        classes = {}
        for i in range(len(Y)):
            try:
                (classes[Y[i]]).append(X[i])
            except KeyError:
                classes[Y[i]] = [ X[i] ]
        if len(classes.keys())!=2:
            raise ValueError( "dataset must have only two classes -
>"+str(len(classes.keys())) )

        features = []
        first_class = classes[classes.keys()[0]]
        second_class = classes[classes.keys()[1]]
        n = len(X[0])
        for i in range(n):
            tp, fp = 0, 0
            for e in first_class:
                if e[i]>0:
                    tp+=1
            for e in second_class:
                if e[i]>0:
                    fp+=1
            tpr = (tp+0.0)/len(first_class)
```

```
             fpr = (fp+0.0)/len(second_class)
                         tpr += (0.0005 if tpr==0 else 0)
                         tpr -= (0.0005 if tpr==1 else 0)
                         fpr += (0.0005 if fpr==0 else 0)
                         fpr -= (0.0005 if fpr==1 else 0)
                         if tpr==0 or tpr==1 or fpr==1 or fpr==0: continue
                         score = abs(self.__ltqnorm(tpr) - self.__ltqnorm(fpr))
                         features.append(Feature(i, score))

                     return features

    def __ltqnorm(self, p ):

        if p <= 0 or p >= 1:
            # The original perl code exits here, we'll throw an exception
instead
            raise ValueError( "Argument to ltqnorm %f must be in open interval
(0,1)" % p )

        # Coefficients in rational approximations.
        a = (-3.969683028665376e+01,  2.209460984245205e+02, \
             -2.759285104469687e+02,  1.383577518672690e+02, \
             -3.066479806614716e+01,  2.506628277459239e+00)
        b = (-5.447609879822406e+01,  1.615858368580409e+02, \
             -1.556989798598866e+02,  6.680131188771972e+01, \
             -1.328068155288572e+01 )
        c = (-7.784894002430293e-03, -3.223964580411365e-01, \
             -2.400758277161838e+00, -2.549732539343734e+00, \
              4.374664141464968e+00,  2.938163982698783e+00)
        d = ( 7.784695709041462e-03,  3.224671290700398e-01, \
              2.445134137142996e+00,  3.754408661907416e+00)

        # Define break-points.
        plow  = 0.02425
        phigh = 1 - plow

        # Rational approximation for lower region:
        if p < plow:
            q  = math.sqrt(-2*math.log(p))
            return (((((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5]) / \
                   ((((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1)

        # Rational approximation for upper region:
        if phigh < p:
            q  = math.sqrt(-2*math.log(1-p))
            return -(((((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5]) / \
                    ((((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1)

        # Rational approximation for central region:
        q = p - 0.5
        r = q*q
        return (((((a[0]*r+a[1])*r+a[2])*r+a[3])*r+a[4])*r+a[5])*q / \
               (((((b[0]*r+b[1])*r+b[2])*r+b[3])*r+b[4])*r+1)
```

```
class Feature:
    def __init__(self, index, score):
        self.index = index
        self.score = score

    def __cmp__(self, other):
        return cmp(self.score, other.score)
```

Listado 7. Clase abstracta para métodos de selección de feature.

```
'''
Created on 5/01/2013

@author: Jorge
'''

class FeatureSelector(object):
    '''
    classdocs
    '''

    def __init__(self):
        '''
        Constructor
        '''
        self.indexes = None

    def get_index_features(self, X, Y):
        raise NotImplementedError( "get_index_features method not implemented" )

    def calculate_best_features(self, X,Y):
        self.indexes = self.get_index_features(X,Y)

    def apply_best_features(self, X):
        new_X = []
        for e in X:
            new_X.append( [f for index,f in enumerate(e) if index in
self.indexes ] )
        return new_X
```

Listado 8. Selección de features usando Rand-Roubin y BNS.

```
'''
Created on 9/03/2013

@author: Jorge
'''
from BinormalSeparation import BinormalSeparation
from FeatureSelector import FeatureSelector
import random

class BinormalSeparationWithRandRoubin(FeatureSelector):
    '''
    classdocs
    '''

    def __init__(self,num_features):
        '''
        Constructor
        '''
        self.num_features = num_features

    def set_num_features(self, num_features):
        self.num_features = num_features

    def get_index_features(self, X, Y):
        indexes_features = []
        classes = self.__create_group(X, Y)
        c = classes.keys()
        features_per_class = []
        prob_per_clas = []
        featureSelector = BinormalSeparation()
        for current in c:
            rest_class = classes.keys()
            rest_class.remove(current)
            rest = []
            for list_claims in [classes[t] for t in rest_class ]:
                for x in list_claims:
                    rest.append(x)

            one_vs_all_X =  [x for x in classes[current] ] + rest
            one_vs_all_Y = [current]*len(classes[current]) + ['rest']*len(rest)
            features = featureSelector.get_feature_score(one_vs_all_X,
one_vs_all_Y)
            features.sort()
            features_per_class.append( features )
            prob_per_clas.append( 1.0/len(classes[current]) )

        prob_per_clas = [prob/sum(prob_per_clas) for prob in prob_per_clas ]
        interval = prob_per_clas
        for i in range(1, len(interval)):
            interval[i] += interval[i-1]
```

```
            k=0
             while k< self.num_features:
                 k+=1
                 numRandom = random.random()
                 for i in range(len(interval)):
                     if (0 if i==0 else interval[i-1]) <= numRandom <=
    interval[i] :
                         temp = None
                         flag=True
                         while flag==True or temp in indexes_features:
                             if len(features_per_class[i])==0:
                                 prob_per_clas.pop(i)
                                 prob_per_clas = [prob/sum(prob_per_clas) for
    prob in prob_per_clas ]
                                 interval = prob_per_clas
                                 for i in range(1, len(interval)):
                                     interval[i] += interval[i-1]
                                 k-=1
                                 temp = None
                                 break
                             else:
                                 temp = features_per_class[i].pop().index
                                 flag=False
                         if temp!=None:
                             indexes_features.append(temp)
                         break

             return indexes_features

def __create_group(self, X,Y):
    m = len(Y)
    group = {}
    for i in range(m):
        try:
            group[Y[i]].append(X[i])
        except KeyError:
            group[Y[i]] = [X[i]]
    return group
```

Listado 9. Clase abstracta para métodos de transformación de datos

```
'''
Created on 4/03/2013

@author: Jorge
'''

class DatasetTransformation(object):
    '''
    classdocs
    '''

    def apply_transformation(self, X):
        return self.transformation(X)

    def transformation(self, X):
        raise NotImplementedError( "transformation method not implemented" )

    def __get_X_set(self, training_set, test_set):
        l_training = len(training_set)
        l_test = len(test_set)
        temp = []
        temp.extend( [e.get_vector_X() for e in training_set])
        temp.extend([e.get_vector_X() for e in test_set])
        return temp, (l_training, l_test)
```

Listado 10. Normalización de datos.

```
'''
Created on 4/03/2013

@author: Jorge
'''
from DatasetTransformation import DatasetTransformation
import numpy as np
from sklearn import preprocessing

class NormalizeData(DatasetTransformation):

    def transformation(self, X):
        n = len(X[0])
        m = len(X)
        mean=[0]*n
        standar_desviation=[0]*n
        for i in xrange(n):
            column_i = [ d[i] for d in X ]
            mean[i]=np.average(column_i)
            standar_desviation[i]=np.std(column_i)

        for i in range(m):
            for j in range(n):
                #print mean[j], standar_desviation[j]
                X[i][j]= (X[i][j]-mean[j])/standar_desviation[j]
                if np.math.isnan( X[i][j]):
                    X[i][j]=0

        return X
```

Listado 11. Transformación TF-IDF

```
'''
Created on 4/03/2013

@author: Jorge
'''
from DatasetTransformation import DatasetTransformation
from sklearn.feature_extraction.text import TfidfTransformer

class TF_IDF(DatasetTransformation):

    def transformation(self, X):
        transformer = TfidfTransformer()
        tfidf = transformer.fit_transform(X)
        new_X = tfidf.toarray()
        return new_X
```

Listado 12. Clase abstracta que implementan todos los clasificadores

```
'''
Created on 7/01/2013

@author: Jorge
'''

from sklearn.grid_search import GridSearchCV
from sklearn.metrics import f1_score, classification_report
from dataset.DatasetImplementations import CustomDataSet

class Classifier(object):

    def __init__(self):
        self.__feature_selector = None
        self.transformation = []
        self.featureSelector = None

    def train (self, X, Y):
        raise NotImplementedError( "train method not implemented" )

    def test(self, X, Y):
        raise NotImplementedError( "test method not implemented" )

    def _get_X_Y_upper_Y_vectors(self, dataset ):
        X=[]
        Y=[]
        for e in dataset:
            X.append(e.get_vector_X())
            Y.append( e.get_numerical_category() )
        return X, Y


    def get_optimal_model(self, classifier, tuned_parameters,X, Y):
        clf = GridSearchCV(classifier, tuned_parameters, score_func=f1_score)
        clf.fit(X, Y, cv=5)
        print clf.best_estimator_
        print classification_report(Y, clf.predict(X))
        return clf.best_estimator_

    def run_train_and_test(self):
        data = CustomDataSet()
        training_set = data.get_training_set()
        #training_set.extend(data.get_validation_set())
        test_set = data.get_test_set()
        l_training = len(training_set)
        l_test = len(test_set)

        X_train, Y_train = self._get_X_Y_upper_Y_vectors(training_set)
        X_test, Y_test = self._get_X_Y_upper_Y_vectors(test_set)
```

```
      if self.featureSelector!=None:
                  #no las calcula. usa las feature obtenidas con validation set
                  self.featureSelector.calculate_best_features(X_train,Y_train)
                  X_train = self.featureSelector.apply_best_features(X_train)
                  X_test = self.featureSelector.apply_best_features(X_test)

              for t in self.transformation:
                  X = t.apply_transformation(X_train+X_test)
                  X_test = X[l_training:l_training+l_test]
                  X_train = t.apply_transformation(X_train)

              self.train(X_train,Y_train)
              self.test(X_test, Y_test)

    def run_train_and_validation(self):
        data = CustomDataSet()
        training_set = data.get_training_set()
        validation_set = data.get_validation_set()
        l_training = len(training_set)
        l_validation = len(validation_set)

        X_train, Y_train = self._get_X_Y_upper_Y_vectors(training_set)
        X_validation, Y_validation =
self._get_X_Y_upper_Y_vectors(validation_set)

        if self.featureSelector!=None:
            self.featureSelector.calculate_best_features(X_train,Y_train)
            X_train = self.featureSelector.apply_best_features(X_train)
            X_validation =
self.featureSelector.apply_best_features(X_validation)

        for t in self.transformation:
            X = t.apply_transformation(X_train+X_validation)
            X_validation = X[l_training:l_training+l_validation]
            X_train = t.apply_transformation(X_train)

        self.train( X_train, Y_train)
        self.test( X_validation, Y_validation)


    def add_transformation_to_data(self, transformation):
        self.transformation.append(transformation)

    def setFeatureSelector(self, featureSelector):
        self.featureSelector = featureSelector

    def optimal_num_feature(self, name):
            data = CustomDataSet()
            training_set = data.get_training_set()
            validation_set = data.get_validation_set()
            l_training = len(training_set)
            l_validation = len(validation_set)

            X_train, Y_train = self._get_X_Y_upper_Y_vectors(training_set)
```

```
X_train, Y_train = self._get_X_Y_upper_Y_vectors(training_set)
            X_validation, Y_validation =
self._get_X_Y_upper_Y_vectors(validation_set)
            num_features = range(100,4600,100)
            output = open('../data/'+name+'.dat','w')
            output.write(name+"\n")
            output.write('features\tf1\n')
            max_f1=0
            best_indexes=None
            best_num_features = None
            if self.featureSelector!=None:
                for num in num_features:
                    output.write(str(num))
                    for iter in range(5):
                        print "num_feature: ",num
                        self.featureSelector.set_num_features(num)

self.featureSelector.calculate_best_features(X_train,Y_train)
                        new_X_train =
self.featureSelector.apply_best_features(X_train)
                        new_X_validation =
self.featureSelector.apply_best_features(X_validation)
                        for t in self.transformation:
                            new_X_validation =
t.apply_transformation(new_X_validation+new_X_train)[0:l_validation
]
                            new_X_train =
t.apply_transformation(new_X_train)

                        self.train(new_X_train , Y_train)
                        f1 = self.test(new_X_validation,
Y_validation)
                        if f1>max_f1:
                            max_f1 = f1
                            best_indexes =
self.featureSelector.indexes
                            best_num_features = num
                        output.write('\t'+str(f1))
                    output.write('\n')
                output.write("mejor ->
f1="+str(max_f1)+"\tnum_features="+str(best_num_features)+"\tindexe
s="+str(best_indexes)+"\n")
            output.close()
```

Listado 13. Bernoulli Naive Bayes.

```
'''
Created on 11/01/2013

@author: Jorge
'''
from Classifier import Classifier
import numpy as np
#from classifiers.svm_tree import SVMTree
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score, confusion_matrix
from DatasetTransformation.TF_IDF import TF_IDF
from feature_selection.BinormalSeparationWithRandRoubin import
BinormalSeparationWithRandRoubin
from feature_selection.OptimalFeatureForClassifiers import
FeatureForMultinomialNB

class MultinomialNaiveBayes(Classifier):

    def train(self, X, Y):
        x = np.array(X)
        y = np.array(Y)
        tuned_parameters = [{
                             'alpha': np.array(range(1,11,1))/10.0,
                            },
                           ]
        self.model = self.get_optimal_model(MultinomialNB(),tuned_parameters,
x,y);
        self.model.fit(x, y)

    def test(self, X, Y):
        x = np.array(X)
        predictions  = self.model.predict(x)
        f1 = f1_score(Y, predictions)
        print ' f1: ',f1
        print ' confusion matrix \n' , confusion_matrix(Y, predictions)
        print classification_report(Y, predictions)
        return f1

if __name__ == '__main__':
    classifier = MultinomialNaiveBayes()
    #classifier.add_tqwransformation_to_data(TF_IDF())
    #classifier.add_transformation_to_data(NormalizeData())
    #classifier.setFeatureSelector(BinormalSeparationWithRandRoubin(800))
    #classifier.setFeatureSelector(FeatureForMultinomialNB())
    classifier.run_train_and_test()
    #classifier.run_train_and_validation()
    #classifier.optimal_num_feature("multinomialNB_rest")
```

Listado 14. Bernoulli Naive Bayes

```
'''
Created on 11/01/2013

@author: Jorge
'''
from Classifier import Classifier
import numpy as np
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score, confusion_matrix
from DatasetTransformation.TF_IDF import TF_IDF
from feature_selection.OptimalFeatureForClassifiers import
FeaturesForBernoulliNB

from feature_selection.BinormalSeparationWithRandRoubin import
BinormalSeparationWithRandRoubin

class BernoulliNaiveBayes(Classifier):

    def train(self, X, Y):
        x = np.array(X)
        y = np.array(Y)
        tuned_parameters = [{
                             'alpha': np.array(range(1,11,1))/10.0,
                            },
                           ]
        self.model = self.get_optimal_model(BernoulliNB(),tuned_parameters,
x,y);
        self.model.fit(x, y)

    def test(self, X, Y):
        x = np.array(X)
        predictions  = self.model.predict(x)
        f1 = f1_score(Y, predictions)
        print ' f1: ',f1
        print ' confusion matrix \n' , confusion_matrix(Y, predictions)
        print classification_report(Y, predictions)
        return f1

if __name__ == '__main__':
    classifier = BernoulliNaiveBayes()
    #classifier.add_transformation_to_data(TF_IDF())
    #classifier.add_transformation_to_data(NormalizeData())
    #classifier.setFeatureSelector(BinormalSeparationWithRandRoubin(1800))
    #classifier.setFeatureSelector(FeaturesForBernoulliNB())
    classifier.run_train_and_test()
    #classifier.run_train_and_validation()
    #classifier.optimal_num_feature("bernoulliNB_rest")
```

Listado 15. SVM con kernel lineal.

```
'''
Created on 12/01/2013

@author: Jorge
'''
from Classifier import Classifier
from sklearn import svm
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxint)
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report, f1_score,confusion_matrix
from DatasetTransformation.TF_IDF import TF_IDF
from DatasetTransformation.NormalizeData import NormalizeData
from feature_selection.BinormalSeparationWithRandRoubin import BinormalSepara-
tionWithRandRoubin
from feature_selection.OptimalFeatureForClassifiers import FeaturesForSVMLinear

class SCIKYSVM(Classifier):

    def train(self, X, Y):
        print 'training...'
        x = np.array(X)
        y = np.array(Y)
        self.model = self.get_optimal_model(x,y)

    def test(self, X, Y):

        print 'testing...'
        x = np.array(X)
        predictions  = self.model.predict(x)
        f1 = f1_score(Y, predictions)
        print ' f1: ',f1
        print ' confusion matrix \n' , confusion_matrix(Y, predictions)
        print classification_report(Y, predictions)
        return f1

    def get_optimal_model(self, X, Y):
        C = 2.0**np.array(range(-5,16,2))
        tuned_parameters = [{
                             'kernel': ['linear'],
                             'C': C,
                             },
                            ]
        clf = GridSearchCV(svm.SVC(C=1, probability=True), tuned_parameters,
score_func=f1_score)
        clf.fit(X, Y, cv=5)
        print clf.best_estimator_
        print classification_report(Y, clf.predict(X))
        return clf.best_estimator_
```

```
if __name__ == '__main__':
    classifier = SCIKYSVM()
    #classifier.setFeatureSelector(BinormalSeparationWithRandRoubin(700))
    #classifier.setFeatureSelector(FeaturesForSVMLinear())
    classifier.add_transformation_to_data(TF_IDF())
    #classifier.add_transformation_to_data(NormalizeData())
    classifier.run_train_and_test()
    #classifier.run_train_and_validation()
    #classifier.optimal_num_feature("svm_linear_rest")
```

Listado 16. SVM con kernel gaussiano.

```
'''
Created on 12/01/2013

@author: Jorge
'''
from Classifier import Classifier
from sklearn import svm
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxint)
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report, f1_score,confusion_matrix
from DatasetTransformation.TF_IDF import TF_IDF
from DatasetTransformation.NormalizeData import NormalizeData
from feature_selection.BinormalSeparationWithRandRoubin import
BinormalSeparationWithRandRoubin
from feature_selection.OptimalFeatureForClassifiers import
FeatureForSVMGaussiano

class SCIKYSVM(Classifier):

    def train(self, X, Y):
        print 'training...'
        x = np.array(X)
        y = np.array(Y)
        self.model = self.get_optimal_model(x,y)

    def test(self, X, Y):
        print 'testing...'
        x = np.array(X)
        predictions  = self.model.predict(x)
        f1 = f1_score(Y, predictions)
        print ' f1: ',f1
        print ' confusion matrix \n' , confusion_matrix(Y, predictions)
        print classification_report(Y, predictions)
        return f1

    def get_optimal_model(self, X, Y):
        C = 2.0**np.array(range(-5,16,2))
        gamma = 2.0**np.array(range(-15,4,2))
        tuned_parameters = [{
                             'kernel': ['rbf'],
                             'C': C,
                             'gamma': gamma
                             },
                            ]
        clf = GridSearchCV(svm.SVC(C=1, probability=True), tuned_parameters,
score_func=f1_score)
        clf.fit(X, Y, cv=5)
        return clf.best_estimator_
```

```
if __name__ == '__main__':
    classifier = SCIKYSVM()
    #classifier.setFeatureSelector(BinormalSeparationWithRandRoubin(700))
    #classifier.setFeatureSelector(FeatureForSVMGaussiano())
    classifier.add_transformation_to_data(TF_IDF())
    #classifier.add_transformation_to_data(NormalizeData())
    classifier.run_train_and_test()
    #classifier.run_train_and_validation()
    #classifier.optimal_num_feature("svm_linear_rest")
```

Listado 17. Lista de stop words utilizadas.

```
de
la
que
el
en
y
a
los
del
se
las
por
un
para
con
no
una
su
al
lo
como
más
pero
sus
le
ya
o
este
sí
porque
esta
entre
cuando
muy
sin
sobre
también
me
hasta
hay
donde
quien
desde
todo
nos
durante
todos
uno
les
ni
contra
otros
ese
eso
ante
ellos
e
esto
mí
antes
algunos
qué
unos
yo
otro
otras
otra
él
tanto
esa
estos
mucho
quienes
nada
muchos
cual
poco
ella
estar
estas
algunas
algo
nosotros
mi
mis
tú
te
ti
tu
tus
ellas
nosotras
vosotros
vosotras
os
mío
mía
míos
mías
tuyo
tuya
tuyos
tuyas
suyo
suya
suyos
suyas
nuestro
nuestra
nuestros
nuestras
vuestro
vuestra
vuestros
vuestras
esos
esas
estoy
estás
está
estamos
estáis
están
esté
estés
estemos
estéis
estén
estaré
estarás
estará
estaremos
estaréis
estarán
estaría
estarías
estaríamos
estaríais
estarían
estaba
estabas
estábamos
estabais
estaban
estuve
estuviste
estuvo
estuvimos
estuvisteis
estuvieron
estuviera
estuvieras
estuviéramos
estuvierais
estuvieran
estuviese
estuvieses
estuviésemos
estuvieseis
estuviesen
estando
estado
estada
estados
estadas
estad
he
has
ha
hemos
habéis
han
haya
hayas
hayamos
hayáis
hayan
habré
habrás
habrá
habremos
habréis
habrán
habría
habrías
habríamos
habríais
habrían
había
habías
habíamos
habíais
habían
hube
hubiste
hubo
hubimos
hubisteis
hubieron
hubiera
```

hubieras
hubiéramos
hubierais
hubieran
hubiese
hubieses
hubiésemos
hubieseis
hubiesen
habiendo
habido
habida
habidos
habidas
soy
eres
es
somos
sois
son
sea
seas
seamos
seáis
sean
seré
serás
será
seremos
seréis
serán
sería
serías
seríamos
seríais
serían
era
eras
éramos
erais
eran
fui
fuiste
fue
fuimos
fuisteis
fueron
fuera
fueras
fuéramos
fuerais
fueran
fuese
fueses
fuésemos
fueseis
fuesen
siendo
sido
tengo
tienes
tiene
tenemos
tenéis
tienen
tenga
tengas
tengamos
tengáis
tengan
tendré
tendrás
tendrá
tendremos
tendréis
tendrán
tendría
tendrías
tendríamos
tendríais
tendrían
tenía
tenías
teníamos
teníais
tenían
tuve
tuviste
tuvo
tuvimos
tuvisteis
tuvieron
tuviera
tuvieras
tuviéramos
tuvierais
tuvieran
tuviese
tuvieses
tuviésemos
tuvieseis
tuviesen
teniendo
tenido
tenida
tenidos
tenidas
tened