Universidad Michoacana de San Nicolás de Hidalgo

División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica

SISTEMA CLASIFICADOR MULTICLASE Y MULTIETIQUETA DE PREGUNTAS DEL SITIO STACKOVERFLOW

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta

José Rafael Cedeño González

Director de Tesis
Dr. en Ciencias Computacionales Juan José Flores Romero

Co-Director
Dr. en Ciencias Computacionales Mario Graff Guerrero

Morelia, Michoacán. Febrero 2015

SISTEMA CLASIFICADOR MULTICLASE Y MULTIETIQUETA DE PREGUNTAS DEL SITIO STACKOVERFLOW

Los Miembros del Jurado de Examen de Grado aprueban
la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de ***José Rafael Cedeño González***

Dr. Jaime Cerda Jacobo
Presidente del Jurado

Dr. Juan José Flores Romero
Director de Tesis

Dr. Félix Calderón Solorio
Vocal

Dr. José Antonio Camarena Ibarrola
Vocal

Dr. Mario Graff Guerrero
Revisor Externo (INFOTEC)

Dr. Félix Calderón Solorio
Jefe de la División de Estudios de Posgrado
de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas).

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Febrero 2015

Dedicado a Liz, que me aligera los retos y alegra mis días

Resumen

Esta tesis aborda el problema planteado en el concurso Identify Keywords and Tags from Millions of Text Questions presentado por Facebook como parte de su programa de reclutamiento, en el cual, utilizando un conjunto de preguntas que fueron ingresadas en el sitio StackOverflow (una red social de colaboración), se intenta predecir las etiquetas asignadas a estas preguntas. Esta clasificación es tanto multi-clase como multi-etiqueta. Para lograr esta clasificación se propusieron dos sistemas simples: uno basado en la distribución de probabilidad obtenida de un clasificador probabilista, y un sistema basado en un conjunto de 5 clasificadores multi-clase. Se discuten los resultados obtenidos por estos diferentes esquemas de caracterización y clasificación; analizando las métricas de precisión y exhaustividad de cada uno de los sistemas de clasificación. Se aprecian resultados competitivos por parte del sistema basado en conjunto de clasificadores, obteniendo valores F1 entre 0.59 y 0.76, los cuales al ser comparados con los resultados obtenidos en el concurso haría este desarrollo ubicarse alrededor del 60^{vo} percentil.

Palabras clave: clasificación multi-etiqueta, minería de datos, aprendizaje automático

Abstract

This work approaches the text document classification problem using as a platform the contest Identify keywords and tags from millions of text questions, in which, by using data from the StackOverflow website, one tries to predict the tags assigned to questions. This categorization is multi-class and also multi-tag. To achieve it, two simple systems are proposed: one based in the probability distribution obtained by a probabilistic classifier, and a 5-way multi-class classifier system. The results obtained by these classification schemes are discussed, analysing score metrics of each classifier system. Competitive results were obtained by the 5-way classifier system, obtaining F1 scores ranging 0.59 to 0.76, which compared with the results showed in the contest would rank this work around the 60^{th} percentile.

Keywords: multi-tag classification, data mining, machine learning

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Glosario

Bag of Words Modelo de representación de texto. 15, 45

Big Data Es un término para toda colección de conjuntos de datos tan grandes o complejos que se vuelve complicado el procesarlos empleando aplicaciones tradicionales de procesamiento de datos. 13

Comma-Separated Values Los archivos CSV (en español valores separados por comas) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. 51

función de pérdida *hinge* Es una función de pérdida utilizada para entrenar clasificadores, en particular máquinas de soporte vectorial. 7

HyperText Markup Language Lenguaje estándar para crear páginas web. 2, 46

Principal Component Analysis Es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos. 37

Receiver Operating Characteristic En español Característica Operativa del Receptor, es una representación gráfica de la exhaustividad frente a (1 - especifidad) para un sistema clasificador binario según se varía el umbral de discriminación. 38

Stochastic Gradient Descent Es un método de optimización por descenso de gradiente para minimizar una función objetivo que se encuentra escrita como una suma de funciones diferenciables. 30

Support Vector Machine Modelos de aprendizaje supervisado que analizan datos y reconocen patrones. 7, 27

Term Frequency - Inverse Document Frequency Estadístico numérico que refleja que tan importante es una palabra para un documento en una colección o diccionario. 6

valor F1 En análisis estadístico, el valor F1 es una métrica de la precisión de una prueba. 11

Vowpal Wabbit Un sistema de aprendizaje automático. 6

Acrónimos

LSVC Linear Support Vector Machine Classification. 68, 70–72, 75

PS Preprocesamiento Simple. 68, 70–72, 75

PT Preprocesamiento Tradicional. 68, 70, 71

SBDP Sistema Basado en Distribución de Probabilidad. 35, 68, 69, 75, 76, 81

SCC Sistema Clasificador por Conjunto de Clasificadores. 35, 69, 71, 75, 81

SCC-NB Sistema Clasificador por Conjunto Naive Bayes. 35, 73, 75, 76

SCC-NB PS Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Simple. 72

SCC-NB PS LSVC Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Simple y Linear Support Vector Machine Classification. 73

SCC-NB PS-VT Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Simple y Variance Threshold. 72

SCC-NB PT Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Tradicional. 72

SCC-NB PT LSVC Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Tradicional y Linear Support Vector Machine Classification. 72

SCC-NB PT-VT Sistema Clasificador por Conjunto Naive Bayes con Preprocesamiento Tradicional Variance Threshold. 72

SCC-SVC Sistema Clasificador por Conjunto Support Vector Machine Classification. 35, 73–76, 78, 81

SCC-SVC PS LSVC Sistema Clasificador por Conjunto Support Vector Machine Classification con Preprocesamiento Simple y Linear Support Vector Machine Classification. 73

SCC-SVC PS VT Sistema Clasificador por Conjunto Support Vector Machine Classification con Preprocesamiento Simple y Variance Threshold. 73

SCC-SVC PT LSVC Sistema Clasificador por Conjunto Support Vector Machine Classification con Preprocesamiento Tradicional y Linear Support Vector Machine Classification. 73

SCC-SVC PT VT Sistema Clasificador por Conjunto Support Vector Machine Classification con Preprocesamiento Tradicional y Variance Threshold. 73

SVM Support Vector Machine. 70, 71, 73

VT Variance Threshold. 68, 75

Capítulo 1

Introducción

El auge de sitios web de colaboración ha traído consigo la necesidad de ordenar y clasificar la vasta cantidad de entradas, preguntas e ideas que se capturan en ellos todos los días. Esta tarea se ha dejado en manos de los usuarios a través de diferentes mecanismos, como son: directorios, archivos, y etiquetas, las cuales actúan como palabras clave para identificar el tema del documento o medio presentado.

Sin embargo, es deseable que esta tarea de categorización sea automática, ya sea para ayudar al usuario a seleccionar la mejor etiqueta para que su audiencia sea la adecuada, o bien para asegurar la correcta categorización de los datos. Es en estos casos donde el aprendizaje automático, en especial los modelos de clasificación, se convierten en una herramienta valiosa para lograr estos objetivos.

Ciertamente, esta tarea presenta numerosos obstáculos, empezando por la cantidad de información que puede presentarse, considerando que sitios moderadamente exitosos pueden albergar miles de entradas de texto y los sumamente exitosos miles de millones. Además, la categorización de estos datos suele ser sobre infinidad de temas y en ocasiones, con sub-temas incluidos, lo cual vuelve a muchos mecanismos tradicionales de categorización insuficientes.

En esta tesis se describe un problema específico sobre la clasificación de documentos de texto para un sitio web, los modelos utilizados y su rendimiento.

1.1. Definición del Problema

El sistema propuesto en esta tesis resuelve el problema planteado en el concurso Identify Keywords and Tags from Millions of Text Questions, publicado en el sitio web kaggle [Goldbloom, 2013]. El concurso consiste en predecir las etiquetas asignadas a un conjunto de preguntas realizadas en el sitio StackOverflow [Attwood, 2013]. StackOverflow es una red social donde se invita a los a usuarios a plantear y responder preguntas, donde estas preguntas son acerca de tópicos de ciencias computacionales. Los usuarios solicitan información acerca de diferentes aspectos de la ciencia computacional, desde uso de un programa en particular hasta preguntas sobre lenguajes de programación y algoritmos. Cada pregunta posee hasta cinco etiquetas, las cuales son palabras clave que sirven para distinguir a las preguntas y poder organizarlas en temas. El usuario que escribe la pregunta es quien se encarga de asignar las etiquetas.

Para el concurso StackOverflow proporcionó un conjunto de entrenamiento de poco mas de seis millones de preguntas el cual consta de un identificador, el título de la pregunta, el contenido de la pregunta y una lista de etiquetas asignadas a la pregunta. También proporcionaron un conjunto de prueba de 20 mil preguntas y que está conformado de los mismos campos que el conjunto de entrenamiento, exceptuando el campo con la lista de las etiquetas asignadas a las preguntas. La validación de los resultados se realiza llenando la columna de etiquetas del archivo que contiene el conjunto de prueba con las predicciones generadas. Posteriormente se envía el archivo al servidor de kaggle que se encarga de verificar los aciertos y fallos obtenidos. El contenido de las preguntas viene tal cual fue ingresado por el usuario, incluyendo etiquetas HyperText Markup Language (HTML) que auxilian al formato de la pregunta. En algunas ocasiones y en contexto a las preguntas se incluyen caracteres no ASCII. También en ciertas preguntas se muestran segmentos de código en diferentes lenguajes de programación y marcación, así como tablas (con o sin formato) alfanuméricas.

El concurso terminó en Enero de 2014, por lo que no fue posible participar con este desarrollo, pero se continuó con el trabajo debido a que es una oportunidad de explorar el funcionamiento de técnicas comunes y de probar nuevas implementaciones.

1.1.1. Caracterización de las preguntas

El primer problema planteado fue cómo convertir el texto en algo que pudiera ser procesado por un mecanismo de categorización y que además pudiera hacerlo para los seis millones de preguntas a procesar.

Un documento de texto se puede ver como un conjunto de palabras y símbolos ordenados de acuerdo a una gramática y lenguaje específico. Estas palabras o elementos léxicos del documento aportan información sobre el documento y el mensaje que intenta transmitir. Es común que si un documento aborda un tema en específico, muchas palabras que lo componen serán sumamente relacionadas al tema tratado y en ocasiones se repetirán a lo largo del documento. Por tanto, una forma elemental de caracterizar un documento de texto es identificar las palabras que lo componen y medir la frecuencia con la que aparecen en el documento.

Usando esta técnica es posible entonces considerar a cada pregunta del problema como un documento de texto, y a su vez se puede formular a cada pregunta como un vector. Las dimensiones de este vector característico serán las palabras que aparecen en todas las preguntas y las coordenadas del vector corresponden a la frecuencia con la que aparecen cada una de las palabras. Preguntas similares corresponden a vectores cercanos. Como consecuencia estas preguntas pertenecerán a la misma clase, o a clases vecinas. Estos vectores se convertirán en las muestras para entrenar un clasificador.

Es evidente que la caracterización de las preguntas trae consigo un problema complejo ya que en el lenguaje inglés existen al menos 23 mil palabras que se consideran comunes [Loper and Bird, 2002], y si se toma en cuenta que muchas de las preguntas son acerca de temas especializados que poseen un léxico propio el número de palabras crece considerablemente.

1.1.2. Reducción del vocabulario de las preguntas

Para evitar que la caracterización generara vectores enormes se propuso restringir el número de palabras que aparecen en las preguntas. Para esto fue necesario definir que palabras son las que no deben de ser incluidas.

En primera instancia se optó por eliminar todas las palabras consideradas como palabras no significativas (en inglés *stopwords*). Estas palabras son todas aquellas que carecen de relevancia como son artículos, pronombres, preposiciones, etc. No existe una definición exacta de palabra no significativa,

por lo que dependiendo del autor o la librería[1] empleada, esta consideración puede variar.

A diferencia de las palabras no significativas que suelen aparecer de manera frecuente a lo largo de diferentes documentos que pueden no tener elementos en común, hay palabras que suelen ser inusuales, como son los nombres propios, identificadores de variables, expresiones textuales, etc. Estas palabras resultan igualmente innecesarias debido a que agregan dimensiones que afectan en ocasiones a una sola pregunta, lo que ocasiona un aumento de la dimensión desmedido, disminuyendo el rendimiento de los procesos utilizados.

Para poder identificar a una palabra como inusual es necesario poseer un diccionario de palabras comunes tanto del lenguaje empleado como de términos propios al tema de las preguntas. Por ejemplo, para un programador puede ser usual hablar de Java o Python, sin embargo alguien ajeno al tema pudiera preguntarse por que se discute sobre variedades de café y de serpientes al mismo tiempo.

1.1.3. Clasificador

Una vez que se haya determinado qué palabras mantener y cómo caracterizar el texto se necesita un mecanismo para identificar las preguntas con una serie de etiquetas.

Para realizar esta distinción entre las etiquetas se planteó el uso de un clasificador. Los clasificadores son modelos estadísticos que sirven para identificar categorías o clases dentro de una población (siendo estas clases subpoblaciones), al encontrarse con nuevas muestras, el clasificador debe ser capaz de predecir a qué categoría o clase pertenecen. En esta tesis se emplea el término etiqueta como la instancia de una clase asignada a una pregunta, y clase para indicar la categoría. La Sección 2.2 define de manera mas formal el proceso de clasificación.

El inconveniente observado es que la gran mayoría de las estrategias de clasificación han sido diseñadas para modelar relaciones 1:1. Es decir, el modelo de clasificación o clasificador únicamente indica si las muestras pertenecen o no a una clase, sin embargo en este desarrollo es necesario modelar

[1]Se emplea a lo largo de este trabajo el término librería en sustitución de biblioteca, esto debido a que el autor se siente mas cómodo empleando este anglicismo, proveniente de *library*

una relación 1:N, ya que el conjunto de preguntas puede particionarse en un número muy grande de clases diferentes. Además, una pregunta puede poseer hasta cinco etiquetas diferentes, por lo que es necesario identificar de manera correcta no solo una clase, sino cuatro mas. Esto convierte a este trabajo en un problema de clasificación multi-clase y multi-etiqueta.

1.1.4. Dimensión vectorial

Aunque el preproceso ayuda enormemente a reducir características, existe la posibilidad de que la dimensión de los vectores generados siga siendo problemática para el entrenamiento de un clasificador, tanto en rendimiento del clasificador como en el tiempo que tarde en ser entrenado. Se necesita identificar la mejor técnica de reducción de dimensiones que maximice la precisión del clasificador.

1.1.5. Manejo de memoria

Si bien la elección de una estrategia de entrenamiento y de una técnica de reducción de dimensiones pudiera ser meramente trabajo de investigación sobre estos temas, el principal inconveniente proviene del sistema de cómputo sobre el cual se realicen estas operaciones, en especial la cantidad de memoria que posea.

El archivo de entrenamiento en crudo posee 7Gb de información y para poder preprocesarlo se requiere de una estrategia donde los resultados parciales del preprocesamiento sean almacenados en archivos parciales para evitar saturar la memoria. Estos resultado parciales deben ser procesados para generar los vectores que representan a las preguntas, de nuevo procurando que la memoria se mantenga en niveles aceptables para la operación del equipo.

Si se considera que son poco mas de seis millones de preguntas, el entrenamiento del clasificador requiere ser planificado por segmentos, ya que de lo contrario el consumo de memoria se dispara considerablemente. De igual manera las técnicas de reducción de dimensiones suelen utilizar grandes cantidades de memoria lo cual las vuelve sumamente problemáticas.

1.2. Antecedentes y estado del arte

Durante la investigación documental de esta tesis se definieron tres aspectos fundamentales para el desarrollo del sistema de clasificación propuesto: la transformación de los datos, el mecanismo de clasificación a emplear y el rendimiento computacional deseable. En este capítulo se examinan estos aspectos explorados por otros participantes del concurso así como por autores interesados en el tema.

Cabe mencionarse que aunque el tema de clasificación multi-etiqueta sea en este momento vanguardista, existen pocas publicaciones al respecto para el caso de la clasificación de documentos de texto, lo cual dificultó en cierta medida la incorporación de ideas y técnicas a este desarrollo.

1.2.1. Transformación de los datos

Debido a que los datos se encuentran tal y como fueron escritos por los usuarios de StackOverflow, es necesario realizar una transformación de estos para poder ser procesados. Este preprocesamiento se discute en la Sección 2.1

Hong y Fang [Hong and Fang, 2013] generan un vector que representa bolsa de palabras, el cual contiene las frecuencias de cada palabra y aplica Term Frequency - Inverse Document Frequency (frecuencia de término – frecuencia inversa de documento, abreviado TF-IDF), la cual es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección de palabras, con la frecuencia de documento calculada usando todo el conjunto de entrenamiento. Se generan "centroides" para cada etiqueta con estos vectores. Posteriormente, en la evaluación para cada pregunta se genera un vector con TF-IDF y se calcula la distancia entre la pregunta evaluada y cada centroide de etiqueta usando la distancia coseno.

Parikh [Parikh, 2013] empleó el *wrapper* vw-varinfo incluido en el algoritmo de aprendizaje automático Vowpal Wabbit (VW) para la selección de características, el cual expone todas las variables del modelo en forma legible para humanos. Esto les permitió identificar la importancia relativa de las palabras y así poder reducir el tamaño del modelo.

Stanley y Byrne [Stanley and Byrne, 2013] crearon un modelo que aplica la siguiente metodología: por cada título y cuerpo de pregunta el modelo devuelve la activación (término que emplean para indicar la respuesta que genera el modelo a la presencia de una palabra) para cada etiqueta posible. La etiqueta con la mayor activación es la que se asocia con esa pregunta. La

fuerza de asociación entre las palabras usadas en las preguntas y las etiquetas fue calculada usando 2/3 del conjunto de datos. Las predicciones del modelo se basan en la cantidad de co-ocurrencias entre las palabras de las preguntas y las etiquetas usadas para esa pregunta. A su vez, las etiquetas fueron agrupadas usando una base de datos de etiquetas sinónimas. Las predicciones del modelo también se basan en la frecuencia de cada etiqueta, por lo que etiquetas frecuentes tendrán una activación base mayor. Realizaron también la eliminación de *stopwords* pero empleando una medida de entropía basada en el contenido del conjunto de datos y no de una lista arbitraria.

Es posible apreciarse que no existe un consenso en cómo transformar el texto para ser procesado, aunque si hay una tendencia a identificar la presencia de ciertas palabras en los documentos de texto. Por tanto, es deseable que el mecanismo de transformación de los datos a emplearse debe ser capaz de seleccionar aquellas palabras que sean relevantes y así poder definir similitud entre las preguntas a categorizar.

Parte de estas ideas fueron aplicadas en el diseño del preprocesamiento utilizado en este trabajo y se precisan en la Sección 3.1 .

1.2.2. Mecanismo de Clasificación

Como se explica en la Sección 2.2, el proceso de clasificación multi-etiqueta requiere atención especial al no poder ser resuelto de manera directa por un solo clasificador.

En Identifying Tags from Millions of Text Questions [Parikh, 2013] se emplearon clasificadores discriminantes, donde se construye un clasificador para cada etiqueta y después se elige el resultado de las etiquetas mas probables. Se determinaron las 500 etiquetas mas comunes y se generaron clasificadores que las identifican. Para predecir las etiquetas que posee una pregunta Parikh utilizó el Algoritmo 1, el cual nos enseña que por cada pregunta a predecir y por cada clasificador entrenado (líneas 3 y 4) obtiene una predicción, estas predicciones se filtran posteriormente (línea 8) para dejar las predicciones mas relevantes, este filtro utilizó un umbral de 0.1 de valor de salida para elegir una etiqueta.

Para realizar la tarea de entrenamiento, Parikh menciona que la *Support Vector Machine* (máquina de soporte vectorial, abreviado SVM) implementada en VW posee dos funciones de pérdida, optando por la función de pérdida *hinge* que fue la que mejores resultados les generó. Debido a esta elección y a trabajos previos, el entrenamiento del modelo fue hecho utilizando una

Algoritmo 1: Predicción por clasificadores discriminantes

```
1 function predicción_discriminante(preguntas, clasificadores):
      Datos: preguntas, el conjunto de preguntas a predecir,
             clasificadores, un arreglo de clasificadores
      Resultado: predicciones, una lista de predicciones ordenada
2     predicciones ← []
3     for pregunta in preguntas do
4         for i ← 0 to i <500 do
5             predicciones_i ← clasificadores_i.predice(pregunta)
6         end
7     end
      /* Filtra las predicciones menos significativas y
         devuelve las 5 predicciones mas significativas     */
8     predicciones ← filtra(predicciones)
9     return predicciones
```

gran cantidad de preguntas que no fueron etiquetadas con la etiqueta que intenta predecir el clasificador, esto debido a que los clasificadores SVM son sensibles a la proporción de muestras positivas contra negativas.

Parikh indica que las etiquetas mas comunes suelen ser predichas con alta precisión mientras que las etiquetas que no pertenecen al grupo de las 500 mas comunes suelen ser predichas con un sinónimo de las 500 mas comunes. Uno de los problemas que le encuentran a este método es su baja exhaustividad (ver definición en la Sección 2.4), lo que se lo atribuyen a que el algoritmo es muy liberal en su clasificación.

En el artículo Predicting Tags for StackOverflow Posts [Stanley and Byrne, 2013] se describe el desarrollo de un modelo inspirado en un modelo probabilista Bayesiano basado en mecanismos de recuperación de memoria declarativa de un ACT-R. Esta es una arquitectura cognoscitiva la cual define operaciones básicas e irreducibles que utiliza la mente humana. Su hipótesis es que las etiquetas utilizadas en las preguntas están basadas en un historial de etiquetas que se usaron previamente y la fuerza de la asociación entre la etiqueta y las palabras contenidas en el título y el cuerpo de la pregunta. El modelo se encuentra descrito con un conjunto de ecuaciones que sirven para determinar la probabilidad de que un fragmento de memoria sea recuperado, dada la probabilidad a priori de recuperar el fragmento en el contexto actual.

En Predicting Tags for StackOverflow Questions [Schuster et al., 2013] Schuster *et. al.* desarrollaron una estrategia híbrida de clasificación donde se generó un detector de lenguaje de programación; esto debido a que en su análisis observaron que muchas preguntas contienen fragmentos de código y que de manera usual las preguntas fueron etiquetadas con el lenguaje de programación en cuestión.

Para realizar la detección de lenguaje de programación, Schuster identificó una lista de 28 etiquetas que hacen referencia a lenguajes de programación. Posteriormente, ubicó los fragmentos de código en las preguntas y se separó cada elemento textual a nivel de caracteres para toda cadena no alfabética. Luego, se filtran todos los elementos que ocurren menos de 20 veces en el corpus generado y con los elementos restantes se entrenó un clasificador Naive Bayes multinomial que devuelve el lenguaje de programación mas probable dado un fragmento de código.

Como sistema secundario implementaron un clasificador basado en una máquina de soporte vectorial donde se trató el problema como un problema de clasificación binaria que predice para cada etiqueta dada si el documento pertenece a una etiqueta o no. A cada pregunta se le realizó un preprocesamiento que consistió de eliminar el código HTML de la pregunta y se separó en elementos léxicos el título y cuerpo de la pregunta, posteriormente se caracterizó cada pregunta como un vector de seis características las cuales son:

- Título exacto: Si la etiqueta es alguna de las palabras en el título.
- Cuerpo exacto: Si la etiqueta es una de las palabras en el cuerpo.
- Título relajado y cuerpo relajado: Si todos los elementos léxicos que se obtienen a separar etiquetas por guión se encuentran en el título o en el cuerpo. Esto es: si java-jdk es la etiqueta parte del cuerpo sería "tengo un error en java al compilar con el jdk".
- Se obtiene la Información Mutua Puntual (en inglés *Pointwise Mutual Information* o PMI) del Título y PMI del Cuerpo: Se itera sobre todas las palabras en el título y cuerpo y se calcula la suma de todos los valores PMI para un par palabra-etiqueta. PMI se define como: $PMI(t, w) = log\frac{p(t,w)}{p(t)p(w)}$. Schuster evalúa las probabilidad para calcular el PMI usando estimados de máxima similitud del conjunto de entrenamiento. En caso

de no observarse un par palabra-etiqueta el valor de esta característica es de cero.

Esta caracterización de las preguntas la utilizaron en el sistema secundario con el fin de identificar las preguntas usando una SVM convencional.

En el artículo Keyword Extraction and Semantic Tag Prediction [Hong and Fang, 2013] desarrollaron cuatro modelos. El primer modelo se basa en una búsqueda de vecino cercano usando las 500 etiquetas mas comunes. Este modelo para cada etiqueta del conjunto de las 500 mas comunes agrupa todas las preguntas que contengan esta etiqueta.

El siguiente modelo que desarrollaron se basa en la co-ocurrencia de etiqueta con palabra clave. Construyeron una matriz de co-ocurrencia para modelar la probabilidad adjunta de palabra y presencia de etiqueta en una pregunta. Por cada etiqueta estiman $\max_{palabra \in consulta} P(y \mid palabra)$ usando la matriz de co-ocurrencia y se devuelven las k etiquetas mas comunes. Mencionan que este modelo sirve potencialmente como un filtro intermediario para los clasificadores que generaron posteriormente.

Otro modelo que formularon es una extensión al modelo anterior usando máquina de soporte vectorial y k-vecinos cercanos (en inglés *k-Nearest Neighbor* o bien k-NN) ya que observaron que el modelo de co-ocurrencias sufre de parcialidad respecto a las etiquetas menos comunes. El utilizar una máquina de soporte vectorial y k-NN mejoraron el rendimiento en conjuntos de entrenamiento pequeños donde el modelo de co-ocurrencia solía sobre entrenar pero fueron menos efectivos en conjuntos de entrenamiento mas grandes.

Por último Hong [Hong and Fang, 2013] desarrollaron un modelo de búsqueda de vecino cercano difuso. Para limitar el efecto de la alta dimensión vectorial visto en el primer modelo implementaron un algoritmo de búsqueda de vecino cercano (en inglés *Nearest Neighbor Search* o bien NNS) difuso para ubicar la similitud semántica de todas las etiquetas posibles. En conjunto con el modelo de co-ocurrencia lograron acotar el espacio de búsqueda de etiquetas posibles a un conjunto de 10-20 etiquetas candidatas. El algoritmo funciona de la siguiente manera: para una pregunta de prueba se usa el modelo de co-ocurrencia para predecir de 10-20 etiquetas, luego por cada etiqueta candidata se extrae su fila correspondiente de la matriz de co-ocurrencia como "centroide" de etiqueta. Mencionan que para incrementar la relevancia de las palabras clave correspondientes a cada etiqueta se pusieron a cero todos los componentes excepto los 500 principales y se re-normalizaron los centroides. Posteriormente se calcula la distancia coseno entre la pregunta a validar y

cada centroide y se reordenan las etiquetas por similitud. Por último muestra las etiquetas correspondientes a los k-centroides mas cercanos de etiqueta.

En los resultados del artículo Keyword Extraction and Semantic Tag Prediction [Hong and Fang, 2013] muestran que el uso del modelo de co-ocurrencia mejora de manera considerable la precisión de los modelos que usan vecino cercano en especial el modelo difuso y que este desarrollo es de muy bajo costo computacional.

Los modelos investigados presentan diferentes maneras de lograr la identificación de las etiquetas en las preguntas, pero de igual manera existen algunas tendencias. Sobresale el uso de clasificadores SVM y la adecuación de estos clasificadores para poder lograr la clasificación multi-etiqueta, la cual no se encuentra implementada en este modelo de clasificación. También se observa que es deseable procesar el menor número posible de características y de etiquetas, esto por motivos que se discuten en la sección siguiente.

1.2.3. Rendimiento y costo computacional

En [Hong and Fang, 2013] mencionan sin embargo que su sistema de clasificación presenta problemas principalmente por que las 500 etiquetas mas frecuentes corresponden tan solo al 61.8 % de todas las etiquetas usadas en las preguntas y que este modelo es muy costoso debido a la alta dimensión que presenta lo cual limita el tamaño del conjunto de entrenamiento reduciendo su precisión.

En el desarrollo creado en [Schuster et al., 2013] obtuvieron como resultados que su enfoque requirió tan solo 50 ejemplos de entrenamiento para alcanzar su máximo valor F1. Sin embargo decidieron utilizar la totalidad del conjunto de entrenamiento para su maquina de soporte vectorial. Observaron también que su clasificador devolvía muchas etiquetas, por lo que optaron limitar el número de etiquetas devueltas por el clasificador a 3.

Intentaron reducir el número de características pero observaron que esto les reducía su valor F1, en especial si se eliminaba la característica PMI de título.

En su artículo añadieron una discusión de sus resultados donde destacan que su enfoque es lento debido a que para probar una pregunta necesitan pasarla por los mil diferentes clasificadores que entrenaron en su sistema para poder realizar una predicción, pero dado que solo usan 6 características y el modelo es muy simple esto se compensa.

Stanley and Byrne [Stanley and Byrne, 2013] mencionan que su modelo es

eficiente computacionalmente ya que en su implementación solo requirieron 3Gb de memoria para formar las matrices necesarias. Consideran que su aplicación potencial es la de generar un sistema que recomienda etiquetas debido a que este modelo identifica etiquetas que contextualmente no son evidentes.

Del trabajo relacionado se concluye que el rendimiento puede verse afectado severamente si no se controla el número de etiquetas a predecir así como el número de características que se obtengan de la transformación de los datos. Además existe el riesgo de ocupar grandes cantidades de recursos computacionales si no se logran optimizar los elementos que integran al sistema clasificador.

1.3. Objetivos

En el desarrollo de esta tesis se propusieron los siguientes objetivos:

1.3.1. Objetivo general

Generar un sistema que pueda sugerir etiquetas para una pregunta en inglés sobre temas de ciencias computacionales con el propósito de ayudar a los usuarios a que sus preguntas puedan ser vistas y posiblemente contestadas por mayor número de expertos.

1.3.2. Objetivos particulares

- Convertir el conjunto de datos a una serie de vectores de dimensiones que permitan entrenar un clasificador con recursos de hardware aceptables.
- Desarrollar una serie de esquemas de clasificación que sean capaces de procesar el conjunto de datos completo con el menor consumo posible de recursos computacionales.
- Alcanzar una puntuación de precisión y sensibilidad competitiva.

1.3.3. Justificación

Se planteó desarrollar este problema por diferentes razones:

- Son datos reales que a diferencia de otros conjuntos de datos de esta naturaleza se encuentran tal cual fueron escritos por los usuarios del sitio StackOverflow sin que se les haya realizado preprocesamiento alguno. Estas condiciones representan una excelente oportunidad para probar técnicas de preprocesamiento y verificar su efectividad para reducir la dimensión vectorial y aumentar la separación espacial entre los vectores que se generen.

- Al ser poco mas de seis millones de preguntas con 42 mil diferentes etiquetas se convierte en un reto para practicar técnicas de manipulación de *Big Data*, lo cual en sí se convierte en un desafío.

- Es una oportunidad de probar diferentes esquemas de clasificación tanto convencionales como no convencionales así como para probar varios clasificadores ante el problema de la clasificación de documentos de texto.

1.4. Descripción de capítulos

Este documento se encuentra organizado de la siguiente manera. En este primer capítulo se presenta el problema a desarrollar y los diferentes subproblemas que lo componen, se discutieron varias estrategias de solución aportadas por varios participantes del concurso, y se proponen una serie de objetivos para solucionar el problema que plantea el concurso.

El Capítulo 2 examina las técnicas empleadas y su fundamento teórico empezando por la caracterización de documentos, donde se analiza la técnica Bag of Words, la cual permite transformar el texto en vectores a manera de histogramas. Posteriormente se habla del preprocesamiento, la importancia de este para generar relevancia entre los datos y potenciar los resultados de clasificación. Se discute el funcionamiento de un clasificador Naive Bayes sobre el cual se basa el desarrollo presentado. A continuación se ven algunas estrategias para clasificación multi-etiqueta.

En el Capítulo 3 se discuten las diferentes técnicas empleadas para la realización de este trabajo, haciendo énfasis en el preprocesamiento empleado; los retos para almacenar de manera simple los datos generados; las técnicas planteadas para realizar los procesos de entrenamiento y predicción de los clasificadores generados, optimizando el uso de la memoria disponible;

así como las estrategias disponibles para reducir el número de características de los vectores generados.

Para el Capítulo 4 se presentan los resultados obtenidos de los diferentes esquemas de clasificación planteados, una comparación entre estos y entre los resultados obtenidos en la competencia así como un análisis estadístico de estos resultados.

Por último, el Capítulo 5 discute el trabajo futuro disponible para este trabajo así como las conclusiones obtenidas de este desarrollo.

Capítulo 2

Marco teórico

En este capítulo se describen los modelos y técnicas utilizados para conformar el desarrollo de esta tesis. Se abordan una serie de consideraciones para la caracterización de documentos de texto, empezando por como representarlos como vectores característicos, procurando limitar el número de características presentes. Posteriormente se reseña la clasificación, los modelos de clasificación empleados y las heurísticas de clasificación multi-clase y multi-etiqueta que se utilizaron. Finalmente se exponen un par de técnicas de selección de características que se utilizaron con el fin de agilizar y mejorar el proceso de clasificación.

2.1. Caracterización de documentos de texto

La caracterización de documentos de texto son técnicas que permiten convertir documentos de texto arbitrarios en características numéricas útiles para algoritmos de aprendizaje automático. Aunque esta tesis no trata de clasificación de documentos, las técnicas usadas en esa área pueden ser usadas para clasificación de preguntas.

Existen diversas maneras de realizar estas conversiones, pero la empleada para este desarrollo se llama "Bolsa de Palabras" (mejor conocido en inglés como *Bag of Words* o bien *BOW*).

Para poder explicar *BOW* se define a continuación el concepto de bolsa o multi-conjunto.

2.1.1. Multi-conjunto

Un multi-conjunto difiere de un conjunto en que cada miembro del mismo tiene asociada una multiplicidad (un número natural), indicando cuantas veces el elemento es miembro del conjunto.

En teoría de conjuntos, un multi-conjunto se define como el par (A, m) donde A es un conjunto y $m : A \to \mathbf{N}$ es una función de A a $\mathbf{N}$. A se conoce como el conjunto subyacente de elementos. Para cada a de A, la multiplicidad de a es el número $m(a)$ [Scheinerman, 2001].

Es común escribir la función m como un conjunto de pares ordenados $\{(a, m(a)) : a \in A\}$.

Por ejemplo, el multi-conjunto escrito como $\{a, b, b\}$ se define como: $\{(a, 1), (b, 2)\}$, mientras que $\{a, a, b\}$ se define como $\{(a, 2), (b, 1)\}$. Finalmente, el multiconjunto $\{a, b\}$ se define como $\{(a, 1), (b, 1)\}$.

2.1.2. Bolsa de palabras

Bolsa de palabras es un modelo donde un texto (que puede ser una oración o documento) se representa como un multiconjunto de las palabras que lo conforman, ignorando la gramática y el orden de las palabras, pero donde se mantiene la multiplicidad de estas.

Implementación

A continuación se muestra el modelado de un texto empleando *BOW*. Se tienen dos documentos de texto:

```
la minería de datos aborda temas como la clasificación de texto
```

```
la minería de datos es útil para la clasificación de imágenes
```

Utilizando estos documentos se forma un diccionario de la siguiente manera:

```
{"la", "minería", "de", "datos", "aborda", "temas", "como",
"clasificación", "texto", "es", "útil", "para", "imágenes"}
```

Este diccionario posee 13 palabras diferentes, y usando los índices del diccionario, cada documento se representa por un vector de 13 dimensiones:

```
[2, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
[2, 1, 2, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1]
```

Donde cada dimensión indica el número de veces que aparece la entrada de diccionario correspondiente, lo cual corresponde a una representación de histograma de los documentos. La palabra `"la"`, aparece en ambos documentos dos veces, igualmente la palabra `"de"`, aunque lo hacen en diferentes partes del texto no es importante para *BOW* ya que solo interesa la ausencia o presencia de la palabra, podemos ver también que `"es"`, `"útil"`, `"para"` e `"imágenes"` no aparecen en el primer documento por lo que para los valores que corresponden al índice de estas palabras son cero, lo mismo sucede en el segundo documento con las palabras `"aborda"`, `"temas"`, `"como"` y `"texto"`.

Se puede apreciar que entre mas extenso sea el vocabulario de los documentos mayor será el tamaño de los vectores que los describen.

2.1.3. Elementos adicionales del texto

Es muy fácil que la caracterización de texto se vuelva extensa si no se controlan algunos elementos presentes en los documentos. Para controlar y ordenar la caracterización del texto se identificaron las siguientes fuentes de ruido: uso de letras mayúsculas, símbolos en el texto, palabras no significativas, vocabulario del texto así como el uso de palabras con la misma raíz. A continuación se describe por qué se consideran estas situaciones como fuente de ruido.

Uso de letras mayúsculas

Supongamos que tenemos los mismos documentos con ligeros cambios:

```
La minería de datos aborda temas como la clasificación de texto
```

```
La minería de datos es útil para la clasificación de imágenes
```

El diccionario resultante es:

```
{"La", "minería", "de", "datos", "aborda", "temas", "como", "la",
"clasificación", "texto", "es", "útil", "para", "imágenes"}
```

En este caso el algoritmo toma a `"La"` y `"la"` como palabras diferentes, lo cual es correcto pero genera una redundancia innecesaria al incluir dos dimensiones que pueden ser cubiertas por una sola. Es conveniente convertir los documentos a un estilo de uso de mayúsculas para hacerlo homogéneo y evitar esta clase de ruido.

Símbolos de puntuación

De igual manera si los documentos tuvieran la estructura:

```
La minería de datos, aborda temas como la clasificación de texto
```

```
La minería de datos es útil para la clasificación de imágenes
```

Aquí el signo de puntuación haría que `"datos,"` se convierta en una dimensión adicional, puesto que posee un caracter adicional que la hace diferente al resto de las palabras contenidas en los documentos.

Es además de notarse que el conjunto de entrenamiento usado en esta tesis viene en formato HTML, por lo que además del texto de las preguntas se incluyen etiquetas que le dan formato, pero que al ser legibles por el algoritmo *BOW* estas etiquetas también se conviertan en palabras. Esto, además de incrementar el número de dimensiones, ocasiona que además del texto el formato de la pregunta sea parte de las características, lo cual no es conveniente ni se encuentra dentro del alcance de este trabajo.

Palabras no significativas

Las palabras no significativas o *stopwords* son todas las palabras que sirven para dar estructura a un documento de texto aunque no aportan información útil para el proceso de clasificación [Rajaraman and Ullman, 2011]. No existe una lista definitiva de palabras no significativas por lo que varía la consideración de palabra no significativa, dependiendo de la herramienta empleada.

Para el caso de el sistema clasificador desarrollado, se optó por deshacerse de las palabras no significativas definidas por la librería scikitlearn [Pedregosa et al., 2011]. Se considera que estas palabras no aportan información relevante al problema e incrementan la cantidad de dimensiones del mismo.

Vocabulario del texto

Al igual que el uso de mayúsculas en las palabras y la presencia de símbolos en el texto, el vocabulario de los documentos altera la dimensión de los vectores resultantes.

Si por ejemplo, existe una palabra que se usa en un solo documento (como es el caso de el nombre de una variable), esta palabra haría que todos los vectores tengan un valor de cero para esa posición y en un solo vector haya un uno, lo cual vuelve a esa pregunta un caso atípico.

Para evitar esta situación es deseable poseer un diccionario de palabras usuales para la temática de los documentos, para poder eliminar las palabras derivadas de las preguntas.

2.1.4. Stemming

Es el proceso para reducir palabras a su forma canónica o raíz. Las palabras "general", "generalización" y "generalizar" poseen la misma raíz "genera".

Es deseable realizar stemming como parte del preprocesamiento de texto debido a que reduce el número de dimensiones de los vectores al transformar muchas palabras diferentes a una sola. Además en muchas ocasiones al conjugar un verbo o declinar una palabra ésta desaparece al momento de caracterizar el texto, por lo que resulta innecesario mantener dichas estructuras.

2.1.5. Preguntas repetidas

Durante la preparación de los datos se observó que algunas preguntas se encontraban repetidas una o mas veces, lo cual no había sido documentado por parte de la organización del concurso. Sin embargo resultaba problemático al incrementar el número de preguntas a procesar de manera innecesaria. Por tanto se identificaron las preguntas repetidas y se removieron del conjunto de datos (esto se implementó empleando una tabla hash). Con esto se

redujo el número de preguntas de 6 millones 300 mil a poco mas de 3 millones 665 mil preguntas, lo que significó una reducción del 41.82 % del total de datos.

2.2. Clasificación

En aprendizaje automático, la clasificación es el problema de identificar a qué conjunto de categorías (sub-poblaciones) pertenece una nueva observación, bajo el sustento de un conjunto de entrenamiento de datos que contiene observaciones (o instancias) cuyas categorías son conocidas [Alpaydin, 2010]. Su definición formal está dada como: para un conjunto de m ejemplos $(x^j, y^j), j = 1, 2, \ldots, m$ (llamado conjunto de entrenamiento) muestreados de una distribución D, donde $x^j \in R^n$ y $y^j \in \{-1, +1\}$. El i-ésimo componente de x^j, x_i^j se denomina la característica i. La clasificación genera una función $f : R^n \to \{-1, +1\}$ que mapea del espacio de muestras al espacio de clases. Esta función puede ser usada para clasificar muestras adicionales $\{x^n\}$ obtenidas de la misma distribución D.

Un ejemplo de clasificación podría ser el asignar a un correo electrónico en particular la clase "spam" ó "no-spam" o bien el asignar un diagnóstico a un paciente dado al ser descrito por las características observadas del paciente (sexo, presión sanguínea, presencia o ausencia de cierto síntoma, etc.).

En la terminología de aprendizaje automático, se considera a la clasificación como una instancia de aprendizaje supervisado. Esto es, aprendizaje donde un conjunto de entrenamiento de observaciones correctamente identificadas se encuentra disponible. El procedimiento correspondiente no supervisado se conoce como *clustering*, el cual involucra agrupar datos en categorías basadas en alguna medida de similitud inherente o distancia [Jiawei and Kamber, 2001].

Frecuentemente las observaciones individuales se componen de un conjunto de propiedades cuantificables, conocidas como variables explícitas o características. Estas propiedades pueden ser categóricas, ordinales, basadas en números enteros o basadas en números reales. Otros clasificadores funcionan comparando observaciones de observaciones previas usando medidas de similitud o distancia.

La clasificación y el *clustering* son ejemplos del problema de reconocimiento de patrones, el cual es la asignación de una categoría a una serie de muestras, dada la identificación de patrones y regularidades encontrada en

los datos. La clasificación es un caso particular del problema de regresión, el cual es el proceso estadístico para estimar la relación entre variables.

En este capítulo se describen dos tipos de clasificadores, los cuales fueron empleados en el sistema desarrollado.

2.2.1. Clasificador Naive Bayes

Un clasificador Naive[1] Bayes asume que la presencia o ausencia de una característica en particular no está relacionada con la presencia o ausencia de cualquier otra característica para la clase dada. Considera que cada una de las características contribuye independientemente a la probabilidad de que una muestra pertenece a una clase, sin importar la presencia o ausencia de otras características [Manning et al., 2008].

Para algunos tipos de modelos de probabilidad, los clasificadores Naive Bayes pueden ser empleados muy eficientemente en un ambiente de aprendizaje supervisado. En muchas aplicaciones prácticas, la estimación de parámetros para modelos Naive Bayes se usa el método de la máxima similitud; en otras palabras, uno puede trabajar con el modelo Naive Bayes sin aceptar la probabilidad bayesiana o usar cualquier método bayesiano.

Se ha estudiado Naive Bayes de manera extensiva desde la década de 1950 y ha permanecido como un método base para el problema de categorización de texto [Russell et al., 1995]. Con el preprocesamiento adecuado, es competitivo en este dominio contra métodos mas avanzados, incluyendo las máquinas de soporte vectorial. Una ventaja de Naive Bayes es que solo requiere una pequeña cantidad de datos de entrenamiento para estimar los parámetros (media y varianza de las variables) necesarios para la clasificación. Debido a que se asumen variables independientes, solo se requiere determinar la varianza de las variables de cada clase y no la matriz de covarianza completa.

Para demostrar el concepto de clasificación Naive Bayes, se considera el ejemplo de la Figura 2.2.1, donde los objetos pueden ser clasificados como verdes o rojos. La tarea es clasificar casos nuevos conforme lleguen, esto es, decidir a que clase pertenecen, basados en los objetos existentes.

[1]Se emplea *Naive* en lugar de su traducción (ingenuo) debido a que se considera como el nombre propio de este método de clasificación.

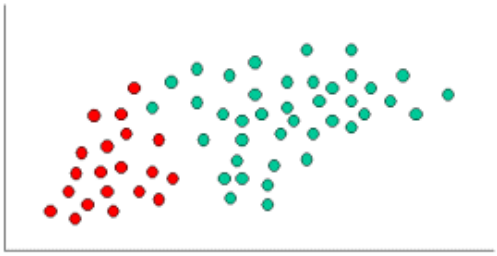

Figura 2.1: Población de objetos verdes y rojos

Debido a que hay mas del doble de objetos verdes que rojos, es razonable creer que un caso nuevo (que no ha sido observado aún) es doblemente mas probable que tenga pertenencia verdes que rojos. En el análisis bayesiano, esta creencia se conoce como probabilidad *a priori*. Las probabilidades a priori se basan en experiencia previa, en este caso el porcentaje de objetos verdes y rojos, y se usa frecuentemente para predecir resultados antes de que sucedan.

Por tanto las probabilidades a priori se pueden escribir como se muestra en las Ecuaciones 2.1 y 2.2:

$$p(x = verdes) = \frac{\text{Número de objetos verdes}}{\text{Número total de Objetos}} \tag{2.1}$$

$$p(x = rojos) = \frac{\text{Número de objetos rojos}}{\text{Número total de Objetos}} \tag{2.2}$$

Como hay un total de 60 objetos, 40 de los cuales son verdes y 20 son rojos, nuestras probabilidades a priori son las indicadas en la Ecuaciones 2.3 y 2.4:

$$p(x = verdes) = \frac{40}{60} \tag{2.3}$$

$$p(x = rojos) = \frac{20}{60} \tag{2.4}$$

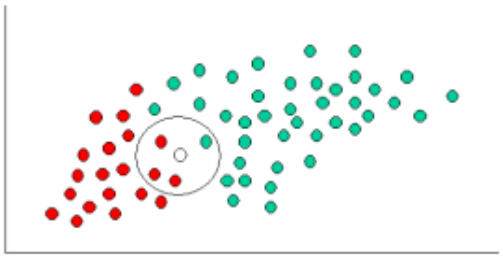

Figura 2.2: Población de objetos verdes y rojos con un elemento recién observado

Habiendo formulado nuestra probabilidad a priori, estamos listos para clasificar un nuevo objeto (círculo blanco) presentado en la Figura 2.2.1. Debido a que los objetos se encuentran bien agrupados es razonable asumir que entre mas objetos verdes (o rojos) existan en la cercanía de X, es mas probable que casos nuevos pertenezcan a ese color en particular. Para medir esta probabilidad, dibujamos un círculo alrededor de x que contenga un número (definido con anterioridad) de puntos sin importar su etiqueta de clase. Posteriormente calculamos el número de puntos en el círculo pertenecientes a cada etiqueta de clase. A partir de esto calculamos la verosimilitud utilizando las Ecuaciones 2.5 y 2.6:

$$L(x \mid verdes) = \frac{\text{Número de verdes cercanos a x}}{\text{Número total de casos verdes}} \tag{2.5}$$

$$L(x \mid rojos) = \frac{\text{Número de rojos cercanos ax}}{\text{Número total de casos rojos}} \tag{2.6}$$

De la Figura 2.2.1, es claro que la similitud de x dado verdes es menor que la probabilidad de x dado rojos, debido a que el círculo agrupa un objeto verdes y tres rojos. La verosimilitud del objeto blanco entre los objetos verdes y rojos queda como se indica en las Ecuaciones 2.7 y 2.8:

$$L(x \mid verdes) = \frac{1}{40} \tag{2.7}$$

$$L(x \mid rojos) = \frac{3}{20} \tag{2.8}$$

Aunque la probabilidad a priori indica que x puede pertenecer a verdes (debido a que hay mas del doble de objetos verdes que rojos) la verosimilitud indica lo contrario; que la clase de x es rojos (dado que hay mas objetos

rojos cercanos a X que verdes). En el análisis bayesiano, la clasificación final es producida combinando ambas fuentes de información. Esto es, la probabilidad a priori y la similitud, para formar una probabilidad posterior usando la llamada regla de Bayes, lo cual se muestra en las ecuaciones 2.9 y 2.10.

$$\begin{aligned} P(verdes \mid x) &= p(x = verdes) \times L(x \mid verdes) \\ &= \frac{4}{6} \times \frac{1}{40} \\ &= \frac{1}{60} \end{aligned} \tag{2.9}$$

$$\begin{aligned} P(rojos \mid x) &= p(x = rojos) \times L(x \mid rojos) \\ &= \frac{2}{6} \times \frac{3}{20} \\ &= \frac{1}{20} \end{aligned} \tag{2.10}$$

Finalmente, se clasifica a x como rojo debido a que su pertenencia de clase alcanza la probabilidad posterior mayor.

Es conveniente mencionarse que aunque este mecanismo de clasificación emplea el teorema de Bayes, no se le considera completamente bayesiano, ya que se asume que las características que componen a las muestras son independientes entre sí [Russell et al., 1995].

El Algoritmo 2 muestra el entrenamiento de un clasificador Naive Bayes. El Algoritmo 4 muestra el proceso de clasificación empleando las probabilidades a priori y una medida de similitud (Algoritmo 3) calculadas por un clasificador Naive Bayes. La notación `[i,:]` indica que la matriz devuelve una sub-matriz a partir de la fila i hasta la fila final.

Como es evidente en el Algoritmo 2, por cada clase distinta en el vector de etiquetas (línea 7) se recuperan las muestras que pertenezcan a esa clase y se cuentan (líneas 8 y 9), se calcula la covarianza y la media entre el número de clases, las muestras y las matrices temporales $\sum_n$ y μ_n con respecto a la i-esima clase (líneas 10) y posteriormente se actualizan las matrices $\sum$ y μ, lo cual se repite para cada clase única. al final, a $\sum$ se le suma ϵ para tener un valor diferente de cero y se calculan las probabilidades a priori (líneas 15 y 16).

Para realizar las predicciones con Naive Bayes, se mencionó que se requiere el cálculo de la similitud entre muestras además de la probabilidad

Algoritmo 2: Entrenamiento de un clasificador Naive Bayes

function `Entrenamiento_Naive_Bayes`(X, y)**:**
  Datos: X matriz de características de la población a clasificar
    y el vector de etiquetas
  Resultado: *priori* Las probabilidades a priori para cada clase
  $\epsilon \leftarrow 1 \times 10^{-9}$
  `/* Se crean dos matrices (`μ `y` $\sum$`) de tamaño` $num\_clases \times num\_car$ `en ceros */`
  $\mu \leftarrow$ `ceros(`num_clases, num_car`)`
  $\sum \leftarrow$ `ceros(`num_clases, num_car`)`
  `/* Se crean dos arreglos (`$priori$ `y` $conteo\_clases$`) de tamaño` num_clases `en ceros */`
  $priori \leftarrow$ `ceros(`num_clases`)`
  $conteo\_clases \leftarrow$ `ceros(`num_clases`)`
  `/* Por cada clases identificada se actualiza la varianza media entre las matrices` θ `y` σ `*/`
  for y_i **in** `distintas(`y`)` **do**
    $X_i \leftarrow X[y == y_i, :]$
    $N_i \leftarrow$ `num_filas(`X_i`)`
    $\mu_n, \sum_n \leftarrow$ `actualiza_var_y_media(`$conteo\_clases_i, \theta[i,:], \sum[i,:], X_i$`)`
    $conteo\_clases_i \leftarrow N_i$
    $\mu[i,:] \leftarrow \theta_n$
    $\sum[i,:] \leftarrow \sigma_n$
  end
  $\sum[:,:] \leftarrow \sum + \epsilon$
  $priori[:,] \leftarrow \frac{conteo\_clases}{\sum_{i=0}^{n} conteo\_clases_i}$
  return *priori*

Algoritmo 3: Cálculo de logverosimilitud conjunta

```
1  function log_verosimilitud_conjunta(X, priori):
       Datos: X Matriz de características de la población a predecir
              priori la lista que contiene las probabilidades a priori de
              cada clase
       Resultado: La log verosimilitud conjunta que posee la matriz X
                  con respecto a la población de entrenamiento
2      lvc ← []
3      for i ← 0 to i < tamaño(clases) do
4          conjunta_i ← ln(priori_i)
5          n_ij ← −0.5× sumatoria(ln(2π ∑[i,:]), eje=0)
6          n_ij ← −(0.5× sumatoria((X_j − θ[i,:])^2 / ∑[i,:], eje=1))
           /* Se agregan los resultados al arreglo lvc        */
7          agrega(lvc, conjunta_i + n_ij)
8      end
9      lvc ← lvc^T
10     return lvc
```

a priori. En el caso particular de esta tesis, se utilizó la log verosimilitud conjunta, la cual es la que tiene implementada la librería empleada y de la cual se muestra su algoritmo (Algoritmo 3).

En el Algoritmo 3 se puede ver que por cada clase (línea 3) se calcula logaritmo natural de la probabilidad a priori de esa clase (línea 4) y se calcula una n_{ij} a la que primero se le suma el negativo de la mitad de la sumatoria del logaritmo natural de la i-esima fila de la matriz $\sum$ multiplicado por 2π (línea 5) y luego le resta la mitad de la sumatoria de las muestras a predecir X menos la i-esima fila de μ al cuadrado sobre la i-esima fila de $\sum$ (línea 6), nótese que la función `sumatoria` posee el parámetro `eje`, el cual sirve para indicar si se van a sumar las filas o las columnas de la matriz en cuestión. Además, obsérvese que si a $\sum$ no se le hubiera sumado ϵ (como lo muestra el Algoritmo 2) se corre el riesgo de una división entre cero (línea 6 del Algoritmo 3 con respecto a la línea 15 del Algoritmo 2). Una vez realizados estos cálculos, se agrega la suma de $conjunta_i$ y n_{ij} al arreglo lvc para posteriormente retornarse como un vector columna (líneas 7 y 9).

Para obtener las predicciones en el modelo Naive Bayes solo se requiere calcular la log verosimilitud conjunta (línea 2) y retornar la etiqueta de la

Algoritmo 4: Predicción a través de un clasificador Naive Bayes

```
1 function Predicción_Naive_Bayes(X, y, num_car, num_clases):
      Datos: X matriz de características de la población a predecir
             priori la lista que contiene las probabilidades a priori de
             cada clase
             clases las etiquetas de las clases
      Resultado: La etiqueta de la clase predicha
      /* Se calculan las log verosimilitudes conjuntas por
         cada x ∈ X con respecto de las probabilidades a
         priori                                              */
2     lvc ← log_verosimilitud_conjunta(X, priori)
      /* Se devuelve la etiqueta de la clase que tenga la
         mayor log verosimilitud conjunta                    */
3     return clases[argmax((lvc))]
```

clase que tenga el mayor valor de log verosimilitud conjunta (línea 3).

2.2.2. Clasificador SVM

Una máquina de soporte vectorial (en inglés *Support Vector Machine*[2]) es un modelo de aprendizaje supervisado que analiza datos y reconoce patrones, lo cual puede ser empleado para clasificar. Un modelo SVM es la representación de las muestras como puntos en un espacio, mapeados de tal manera que las muestras de las diferentes categorías se encuentren particionados por un margen claro que sea tan ancho como sea posible. Muestras nuevas se mapean en el mismo espacio y se predice su pertenencia a una categoría basándose en que lado del margen se llegan a ubicar.

Las máquinas de soporte vectorial se basan en el concepto de planos de decisión que definen fronteras de decisión. Un plano de decisión es aquel que separa entre un conjunto de objetos que poseen diferentes asignaciones de clase. En el siguiente ejemplo, demostrado en la figura se los objetos pueden pertenecer a la clase verdes o rojos. La línea separadora define una frontera en el lado derecho de la cual todos los objetos son verdes y a la izquierda donde todos los objetos son rojos. Cualquier objeto nuevo que caiga hacia

[2]Se usa el acrónimo SVM a lo largo de este trabajo por que es el término por el que se le conoce en el área de clasificación

el lado derecho será etiquetado (clasificado) como verdes (o clasificado como rojos si cayera del lado izquierdo de la línea separadora).

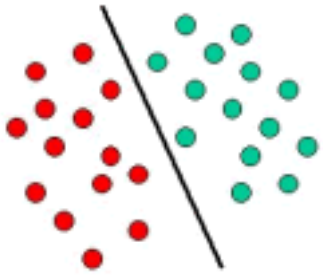

Figura 2.3: Población de objetos verdes y rojos separados por una línea recta

El ejemplo mostrado en la Figura 2.2.2 representa un clasificador lineal, esto es, un clasificador que separa un conjunto de objetos en sus grupos respectivos (verdes y rojos en este caso) con una línea. Sin embargo, muchas tareas de clasificación no son tan simples, y muchas veces se requieren estructuras complejas para generar una separación óptima. Esta situación se ilustra en la siguiente imagen. Comparado con el esquema anterior, está claro que una separación completa de los objetos verdes y rojos pudiera requerir una curva (que es mucho mas compleja que una línea), como se observa en la Figura 2.2.2. Las tareas de clasificación basadas en dibujar líneas separadoras para distinguir entre objetos de diferentes clases se conocen como clasificadores de hiperplanos. Las máquinas de soporte vectorial son particularmente aptas para manejar estas tareas.

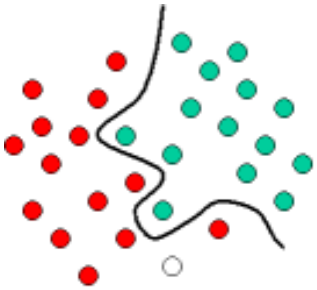

Figura 2.4: Población de objetos verdes y rojos separados con una línea no recta

La Figura 2.2.2 muestra la idea básica detrás de las máquinas de soporte vectorial. Se puede observar los objetos originales (del lado izquierdo del esquema) mapeados, esto es, reordenados, empleando un conjunto de funciones matemáticas conocidas como kernels, los cuales se encargan de realizar la conversión del espacio de entrada al espacio de características. El proceso de reordenar los objetos es conocido como mapeo (transformación). Es de notarse que en este nuevo marco de referencia, los objetos mapeados (al lado derecho de la imagen) son linealmente separables y por tanto, en vez de construir una curva compleja (lado izquierdo), lo único que se debe hacer es encontrar la línea que separe de los objetos verdes y rojos de manera óptima.

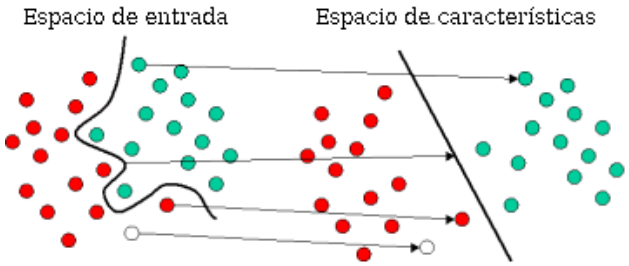

Figura 2.5: Mapeo de un espacio de entrada a un espacio de características

Para construir un hiperplano óptimo, las SVM emplean un algoritmo de entrenamiento iterativo, el cual se usa para minimizar una función de error. De acuerdo a la forma de la función de error, los modelos SVM pueden ser clasificados como: SVM de clasificación Tipo 1, SVM de clasificación Tipo 2 y Descenso por gradiente estocástico.

A continuación se describen las SVM de clasificación Tipo 1, 2 y de descenso por gradiente estocástico.

SVM de clasificación Tipo 1

Para este tipo de SVM, el entrenamiento utiliza la función de minimización de error 2.11:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i \tag{2.11}$$

Sujeto a las restricciones 2.12:

$$y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, N \tag{2.12}$$

donde C es la constante de capacidad, w es el vector de coeficientes de ponderación, el cual se usa para dar mayor relevancia a ciertas características por si así se desea, b es una constante, ξ_i representa parámetros para manejar datos no separables (entradas) y ϕ es un método kernel que se usa para transformar los datos de la entrada (independiente) al espacio de características. El índice i etiqueta los N casos de entrenamiento. Obsérvese que $y \in \pm 1$ representa las etiquetas de clase y x_i representa a las variables independientes. Debe notarse que mientras mas grande sea C, el error se penaliza mas, por consiguiente, C debe elegirse con cuidado para evitar sobre-ajuste.

SVM de clasificación Tipo 2

En contraste al tipo de clasificación anterior, el modelo Tipo 2 minimiza la función de error como 2.13:

$$\frac{1}{2}w^T w - v\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i \tag{2.13}$$

Sujeto a las restricciones 2.14:

$$y_i(w^T\phi(x_i + b) \geq \rho - \xi_i \text{y} \xi_i \geq 0, \rho \geq 0, \ldots, N \tag{2.14}$$

Descenso por gradiente estocástico

En este tipo el entrenamiento la función de error se describe en el Algoritmo 5:

El Algoritmo 5 indica que durante un número n de veces, empleando un vector de parámetros w, este vector se va ajustando por descenso de gradiente utilizando una función de pérdida Q_i limitada por una tasa de aprendizaje α. La función Q_i es dependiente del problema. La implementación utilizada hace uso de la función de pérdida *hinge* [Rosasco et al., 2004].

Para el caso que nos compete se utilizó una SVM entrenada utilizando como función de error a descenso por gradiente estocástico (cuyas iniciales en inglés son *Stochastic Gradient Descent*) debido a que la implementación empleada es muy eficiente (hace uso de librerías de álgebra lineal optimizadas) y SGD puede operar en lotes, lo cual es indispensable por la cantidad de datos a procesar.

Algoritmo 5: Descenso por gradiente estocástico

```
1 function descenso_por_gradiente_estocastico(w, α, Q):
      Datos: w vector inicial de parámetros
             α tasa de aprendizaje
             Q la función de pérdida a utilizar
      Resultado: El vector de parámetros ajustado
2     for i = 1, 2, ..., n do
3         w ← w − α∇Q_i(w)
4     end
5     return w
```

2.2.3. Clasificación Multiclase

La clasificación multi-clase se define como el problema que, para un conjunto de m ejemplos $(x^j, y^j), j = 1, 2, \ldots, m$ (llamado conjunto de entrenamiento) muestreados de una distribución D, donde $x^j \in R^n$ y $y^j \in \{0, 1, \ldots, k\}$. El i-esimo componente de x^j, x_i^j se denomina la característica i. La clasificación genera una función $f : R^n \to \{0, 1, \ldots, k\}$ que clasifica elementos adicionales $\{x^n\}$ muestreados de la misma distribución D.

Existen varias estrategias para reducir el problema de clasificación multiclase a múltiples problemas de clasificación binaria. En esta sección se presentan dos de ellas.

One-vs.-rest

La estrategia *One-vs.-rest* [Bishop, 2006] (también conocida como *One-vs.-all*, OvR ú OvA) involucra entrenar un clasificador simple por clase, con las muestras de esa clase como muestras positivas y todas las demás como negativas. Esta estrategia requiere que los clasificadores base produzcan una puntuación de confianza para su decisión, en vez de tan solo una etiqueta de clase.

El algoritmo de entrenamiento para un aprendiz OvR construido a partir de un modelo de clasificación binario es indicado en el Algoritmo 6. Este algoritmo indica que por cada clase conocida (línea 2) entrene un clasificador binario. La función `entrena` se utiliza para indicar que se utiliza el método de entrenamiento adecuado para el modelo elegido (línea 4).

Para generar una predicción se le muestran a todos los clasificadores una

Algoritmo 6: Entrenamiento de clasificadores multi-clase OvR

function `entrenamiento_ovr(`L, X, y`)`**:**
  Datos: L un modelo de clasificación binaria
    X conjunto de muestras
    y etiquetas donde $y_i \in \{1, \ldots, K\}$ es la etiqueta para la muestra X_i
  Resultado: Una lista de clasificadores f_k para $k \in \{1, \ldots, K\}$
  for k **in** $\{1, \ldots, K\}$ **do**
    Construye un nuevo vector de etiquetas $y_i = 1$ donde $y_i = 1, 0$ (o -1) en el resto
    `/* la función entrena indica el método de entrenamiento para el modelo` L `usando la población` X `con el vector de etiquetas` y `*/`
    $f_i \leftarrow$ `entrena(`L`,(`X`,`y`))`
  end

muestra desconocida x y se predice una etiqueta k la cual corresponde a la que posea la puntuación de confianza mas alta (2.15).

$$\hat{y} = \underset{k \in 1,\ldots,K}{\arg\,\max} f_k(x) \tag{2.15}$$

Aunque esta estrategia es popular, es una heurística que sufre muchos problemas. Primero, la escala de los valores de confianza puede diferir entre los clasificadores binarios. Segundo, incluso si la distribución de clases se encuentra balanceada en el conjunto de entrenamiento, los aprendices de clasificación binaria observan distribuciones no balanceadas debido a que típicamente el conjunto de muestras negativas que ven es mucho mas grande que el conjunto de muestras positivas.

One-vs.-one

En la reducción *one-vs.-one* (OvO), se entrenan $K(K-1)/2$ clasificadores binarios para un problema multi-clase con K clases; cada clasificador recibe las muestras de un par de clases del conjunto de entrenamiento original y debe aprender a distinguir entre estas dos clases. Durante la fase de predicción, se aplica un esquema de votación; todos los $K(K-1)/2$ clasificadores reciben

una muestra no observada y la clase que obtenga el mayor numero de votos es la que se predice por el clasificador combinado.

De igual manera que OvR, OvO sufre de ambigüedades en la que algunas regiones de su espacio de entrada pueden recibir el mismo número de votos.

Los clasificadores Naive Bayes son inherentemente multi-clase debido a que por su construcción son capaces de observar mas de dos clases. La implementación del clasificador SVM utilizado en esta tesis hace uso de la estrategia OvR (la documentación de la librería no especifica esta elección).

2.2.4. Clasificación Multi-etiqueta

La clasificación multi-etiqueta es una variante del problema de clasificación donde múltiples etiquetas se asignan a cada instancia. Se puede interpretar como el problema de encontrar un modelo que mapee entradas x a vectores binarios y, en vez de salidas escalares en el problema de clasificación ordinario. Estos vectores binarios indican las múltiples membresías que una muestra posee respecto a las diferentes clases conocidas, donde la posición en el vector representa a la clase y el valor (0 ó 1) indica si pertenece o no a la clase.

Existen dos métodos principales para abordar el tema de clasificación multi-etiqueta: métodos de transformación de problema y métodos de adaptación de algoritmo.

Métodos de transformación de problema

Existen diversos métodos de transformación de problema para el problema de clasificación multi-etiqueta. En la aproximación fundamental, llamada método de relevancia binaria, se entrena de manera independiente un clasificador binario por cada clase conocida. Dada una muestra no observada, el modelo combinado predice todas las etiquetas donde cada uno de los clasificadores binarios generan un resultado positivo para esta muestra. Este método de dividir la tarea en múltiples tareas binarias tiene mucho en común con el método OvR de clasificación multi-clase. Es de notarse que no es el mismo método: en relevancia binaria se entrena un clasificador por cada etiqueta, no un clasificador por cada valor posible de la clase.

Existen otros tipos de transformaciones. De estos la transformación *label powerset* (LP) genera un clasificador binario por cada combinación observada en el conjunto de entrenamiento. El algoritmo *RAKEL* [Tsoumakas and

Vlahavas, 2007] (k-conjuntos de etiqueta aleatorios) usa varios clasificadores LP, cada uno entrenado sobre un subconjunto aleatorio de etiquetas, la predicción emplea un esquema de votación.

Métodos de adaptación de algoritmo

Algunos modelos/algoritmos se han adaptado para el problema multi-etiqueta, sin requerir trasformación de problema. Algunos de estos métodos son:

- k-nn (Vecinos mas cercanos): El algoritmo ML-KNN extiende el clasificador k-nn para datos multi-etiqueta [ling Zhang and hua Zhou, 2007].
- Árboles de decisión: Clare es un algoritmo C4.5 adaptado para clasificación multi-etiqueta; la modificación involucra un cambio en los cálculos de entropía [Madjarov et al., 2012].
- Redes Neuronales. BP-MLL es una adaptación del algoritmo de *back-propagation* para aprendizaje multi-etiqueta [Zhang and Zhou, 2006].

Ni Naive Bayes ni SVM poseen un mecanismo multi-etiqueta formal, sin embargo, para el caso de Naive Bayes, si se considera que para realizar sus predicciones debe calcular la probabilidad que tiene la muestra no observada de pertenecer a cada una de las clases conocidas, se puede entonces tomar esta distribución de probabilidad y en vez de predecir una sola etiqueta se recuperan las n etiquetas con mayor probabilidad.

Para el presente trabajo se plantearon un par de sistemas de clasificación: un sistema que utilice la distribución de probabilidad calculada por un clasificador Naive Bayes con el propósito de recuperar la serie de clases mas probables para una pregunta. Esto es, modificar el comportamiento tradicional de Naive Bayes que se muestra en el algoritmo 4 por el mostrado en la ecuación 2.16, donde las etiquetas a, b y c se obtienen de la primera, segunda y tercera probabilidades mas altas provenientes de la distribución de probabilidad p generada por el modelo, w indica la clase que posee la mayor probabilidad de todas, x la clase con la segunda probabilidad mas alta y z la clase con la tercera probabilidad mas alta para toda clase y conocida. Este sistema fue nombrado Sistema Basado en Distribución de Probabilidad

(SBDP).

$$\begin{aligned}\hat{y} = \{(a,b,c)|a &:= \{w|\forall y : p(y) \leq p(w)\},\\ b &:= \{x|\forall y : p(y) \leq p(x) \leq p(w)\},\\ c &:= \{z|\forall y : p(y) \leq p(z) \leq p(x) \leq p(w)\}\}\end{aligned} \tag{2.16}$$

El sistema complementario es un sistema de clasificación basado en un conjunto de clasificadores donde se entrenan cinco clasificadores, uno por cada etiqueta asignable a cada pregunta. Esto es, de manera semejante a los métodos de transformación de problema demostrados en la ecuación 2.15 pero en vez de realizarlo con clasificadores binarios sea con clasificadores multi-clase. Al ser clasificadores multi-clase cada clasificador es capaz de asignar cualquier etiqueta que haya sido observada durante el entrenamiento. A este esquema se le asignó el nombre de Sistema Clasificador por Conjunto (SCC) el cual tiene dos variantes, SCC-NB (Sistema Clasificador por Conjunto Naive Bayes) y SCC-SVC (Sistema Clasificador por Conjunto Support Vector Machine) las cuales se describen en el siguiente capítulo.

2.3. Selección de características

El número de características que genera el proceso de vectorización de las preguntas llega a ser cuantioso para las implementaciones de los modelos de clasificación presentados, y conjugado con la cantidad de preguntas a procesar vuelve sumamente complicada cualquier operación que se realice sobre los datos. La selección de características es el proceso de seleccionar subconjuntos de características relevantes para emplearlas en la construcción del modelo. En ocasiones sucede que algunas características resultan redundantes o bien irrelevantes para el modelo de clasificación, lo que ocasiona que el modelo de clasificación sea impreciso y lento.

Existe una gran variedad de técnicas de selección de características, muchas de ellas basadas en análisis estadístico. Para este desarrollo se emplearon dos técnicas: umbral de varianza y SVC [3] lineal.

Umbral de varianza (*Variance Threshold* en inglés) fue elegida debido a su simplicidad y bajo costo computacional. Además, la técnica de vectorización

[3]Se emplea el acrónimo de *Support Vector machine for Classification* por simpleza y su reconocimiento en el ámbito del aprendizaje supervisado.

seleccionada, los vectores generados tienen muchas características con valor cero, lo cual ocasiona que la varianza de estas características sea poca y por tanto innecesaria.

SVC lineal fue elegida ya que se ha observado [Guyon and Elisseeff, 2003] que al realizar una ligera modificación al algoritmo SVM se puede entrenar una SVM empleando la norma L_1 como penalización. Se le considera una penalización debido a que mediante esta norma se puede formular una función de optimización que genera un espacio disperso donde es posible eliminar o penalizar a los coeficientes que sean cero.

La familia de norma vectorial L, $L_p : \mathbb{R}^d \to \mathbb{R}$ se define como lo indica la ecuación 2.17:

$$L_p(x) = \|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}} \tag{2.17}$$

donde x es un vector y d son las dimensiones de este.

En base a estas normas vectoriales es posible generar funciones de optimización. Para este caso, la función de optimización empleada se basa en la norma L_1, la cual es descrita en la ecuación 2.18, donde w es un vector de ponderaciones y λ es un parámetro sintonizado empíricamente.

$$\min_w (Y - Xw)^T (Y - Xw) + \lambda \|w\|_1 = \min_w (Y - Xw)^T (Y - Xw) + \lambda \sum_{i=1}^{d} |w_i| \tag{2.18}$$

Esto ocasiona que el espacio obtenido sea disperso y que muchos de los coeficientes sean cero, por lo anterior es posible generar un método que seleccione todos los coeficientes diferentes de cero y por consiguiente reducir el número de dimensiones de los datos.

2.3.1. Umbral de varianza

La técnica umbral de varianza identifica aquellas características que posean poca varianza. Este algoritmo sólo observa a la matriz de características y no a las etiquetas, por lo que puede ser empleado para entrenamiento no supervisado. El algoritmo 7 describe el funcionamiento de esta técnica. Donde, de manera inicial se obtiene la varianza media entre los valores de las características por fila (línea 2), y por cada característica presente (línea 3),

si la varianza de esa característica supera al umbral (línea 4), esta columna de la matriz se mantiene, de lo contrario se deshecha (línea 5). Al final se retorna una nueva matriz de características reducida.

Algoritmo 7: Selección de Características mediante Umbral de Varianza

```
1 function umbral_varianza(X, umbral, num_car):
      Datos: X, la matriz de características
      umbral, un umbral especifico para considerar la varianza como
      pobre
      num_car, el número de características
      Resultado: X_n la matriz de características con las características
                 con menor varianza eliminadas
      /* Se calcula la varianza por columna                    */
2     varianzas ← varianza_media(X, eje=0)
3     for i ← 0 to i < num_car do
4         if umbral < varianzas_i then
5             agrega_columna(X_n, X, i)
6         end
7     end
8     return X_n
```

Aunque existen similitudes de este método con PCA (Análisis de Componentes Principales, en inglés *Principal Component Analysis*), no se utilizó PCA por que la implementación empleada de esta técnica no pudo transformar los datos por exceso de consumo de memoria y limitaciones que posee esta librería con respecto al número de características que pueda tener el conjunto de datos.

2.3.2. SVC Lineal

SVC lineal genera una máquina de soporte vectorial para clasificación, la cual se descarta y se preservan los datos ajustados mediante la penalización L1 y un umbral de convergencia que controla el número de características a preservar. El algoritmo 8 muestra el funcionamiento de esta técnica. Lo que hace el algoritmo es que mientras el valor del error sea mayor que el umbral determinado (línea 3), se entrena una SVC con las muestras X y sus respectivas etiquetas y (línea 4), se recupera el vector w de ponderaciones

Algoritmo 8: svc_lineal(X, y, umbral)

```
1  function svc_lineal(X, y, umbral):
       Datos: X, la matriz de características
       y, lista de etiquetas de cada pregunta
       umbral, un umbral de convergencia para controlar el error y el
       número de características a conservar
       Resultado: X_n la matriz de características ajustada
2      error ← 1 × 10^20
3      while error > umbral do
4          svc ← entrena_SVC(X, y)
5          w ← obten_w(svc)
6          X ← X × w
7          error ← calcula_error(svc, umbral)
8      end
9      X_n ← selecciona_columnas(X)
10     return X_n
```

generado por la SVC y este se multiplica por X (líneas 5 y 6), la SVC al ser entrenada con la penalización L1 va a generar una solución dispersa lo que va a ir haciendo cero las características menos relevantes, por lo que la matriz que devuelve este algoritmo se procesa para descartar las columnas en cero (línea 9) y se devuelve esta nueva matriz de características reducida.

2.4. Métricas de desempeño de clasificación

Las métricas empleadas para medir la efectividad de los mecanismos de clasificación son la precisión, la exhaustividad y el valor F1, ajustados a clasificación multi-clase y multi-etiqueta. No se incluye en esta tesis el análisis de curva *Receiver Operating Characteristic* (ROC) debido a los informes generados por las librerías usadas no contienen la información necesaria para construirlos (los informes solo proporcionaron las métricas descritas en esta sección, mas no los valores para cada evento de clasificación). Aunque es posible calcularlos a partir de las predicciones obtenidas, no se pudo realizar por restricciones de tiempo. En lugar de esto se muestran los promedios de los valores de precisión, exhaustividad y valor F1 por etiqueta, los cuales no son visuales, pero de igual manera permiten medir la efectividad de los

sistemas de clasificación.

2.4.1. Métricas en clasificación binaria

A continuación se definen los diferentes eventos posibles al realizar predicciones y posteriormente se describe las métricas utilizadas, empezando primero para el caso de clasificación binaria y posteriormente su uso en el esquema de clasificación multi-clase.

Eventos de clasificación

Al momento de realizar predicciones con un clasificador pueden suceder cuatro casos diferentes los cuales son: verdadero positivo, falso positivo, verdadero negativo y falso negativo.

Se considera que un resultado es verdadero positivo cuando el clasificador predice acertadamente que una muestra pertenece a una clase y efectivamente lo es. Falso positivo es cuando el clasificador identifica una muestra con una clase y en realidad no lo es. Verdadero negativo sucede cuando el clasificador indica que una muestra no pertenece a cierta clase y de hecho no lo es. Falso negativo es cuando un clasificador predice que una muestra no pertenece a una clase cuando verdaderamente si pertenece a esa clase. La Figura 2.4.1 muestra un ejemplo de estos sucesos.

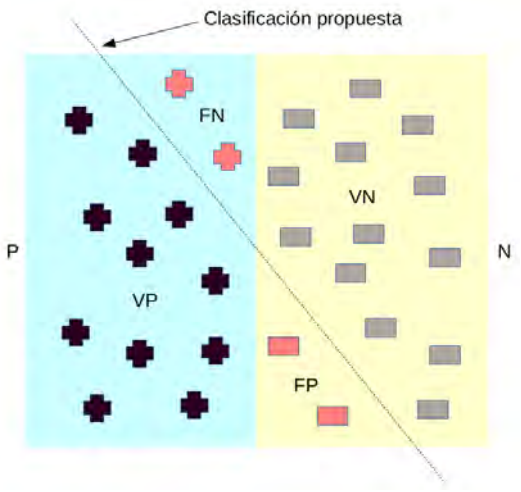

Figura 2.6: Ejemplo de eventos de clasificación

Tanto los falsos positivos como los falsos negativos se consideran errores de clasificación, denominados error tipo 1 y error tipo 2 respectivamente. Las probabilidades de que sucedan estos errores se denominan α y β repectivamente.

Empleando estos sucesos se puede medir la efectividad de un clasificador.

Precisión binaria

La precisión o valor predictivo positivo se define dentro de la clasificación binaria como la proporción de resultados positivos que son verdaderos positivos. Se expresa como:

$$p = \frac{\text{número de verdaderos positivos}}{\text{número de verdaderos positivos} + \text{número de falsos positivos}} \tag{2.19}$$

Aunque se pudiera considerar la precisión como suficiente para medir la efectividad de un clasificador, esta métrica no considera los errores tipo 2. Por tanto es conveniente incorporar la siguiente métrica.

Exhaustividad en clasificación binaria

La exhaustividad o proporción de verdaderos positivos (también conocida como sensibilidad — en inglés como *recall rate* o simplemente *recall*) mide la proporción de positivos existentes que son identificados correctamente como tales.

$$r = \frac{\text{número de verdaderos positivos}}{\text{número de verdaderos positivos} + \text{número de falsos negativos}} \tag{2.20}$$

Utilizando en conjunto la precisión y la exhaustividad es posible medir la efectividad de un clasificador para identificar adecuadamente tanto los eventos positivos como negativos sin que sea muy liberal (diga que todos son positivos o bien negativos) ni muy estricto (que considere a muy pocos como positivos o bien muy pocos como negativos). Sin embargo para poder medir la relación entre estas métricas se emplea el valor F1.

Valor F1

El valor F1 [Rijsbergen, 1979] se expresa como lo muestra la Ecuación 2.21.

$$F1 = 2\frac{p \times r}{p + r} \tag{2.21}$$

donde p es la precisión y r es la exhaustividad. El valor F1 puede ser interpretado como la media armónica de la precisión y la exhaustividad donde el valor es mejor cuando alcanza uno y peor cuando tiende o es cero. Se emplea esta métrica en el problema de clasificación ya que pondera de igual manera tanto a la precisión como a la exhaustividad, las cuales un buen clasificador intentará maximizar de manera simultanea, lo cual significa que un clasificador, además de ser preciso, es bueno delimitando sus predicciones.

2.4.2. Cálculo de métricas en clasificación multi-clase

En el caso de clasificación multi-clase el cálculo de estas métricas se realiza utilizando la media ponderada de cada uno de los resultados por etiqueta, para todas las etiquetas del problema, M.

Precisión multi-clase

El cálculo de la precisión en el caso multi-clase se muestra en la Ecuación 2.22.

$$p_{multi} = \sum_{i=1}^{N} \frac{p_i s_i}{n} \tag{2.22}$$

donde, p_i es la precisión del clasificador obtenida para la clase i, s_i representa el número de muestras que existen en el conjunto de prueba que pertenecen a la clase i, n es número total de muestras contenidas en el conjunto de prueba y N es el total de clases que conoce el clasificador.

Exhaustividad multi-clase

Para calcular la exhaustividad en la clasificación multi-clase se emplea la Ecuación 2.23.

$$r_{multi} = \sum_{i=1}^{N} \frac{r_i s_i}{n} \tag{2.23}$$

donde, la variable r_i representa la exhaustividad que posee un clasificador para la clase i, s_i es el número de muestras contenidas en el conjunto de prueba que pertenecen a la clase i, n es el total de muestras que posee el conjunto de prueba y N es el número total de clases que observó el clasificador.

Valor F1 multi-clase

El valor F1 en clasificación multi-clase se define en la Ecuación 2.24.

$$F1_{multi} = \sum_{i=1}^{N} \frac{F1_i s_i}{n} \tag{2.24}$$

donde $F1_i$ es el valor F1 calculado para la clase i, s_i es el número de muestras contenidas en el conjunto de prueba que pertenecen a la clase i, n es el número total de muestras contenidas en el conjunto de prueba y N es la cantidad de clases que observó el clasificador.

2.4.3. Cálculo de métricas en clasificación multi-etiqueta

Para calcular los valores de estas métricas para problemas multi-etiqueta, se calcula la media de cada clasificador por etiqueta, utilizando para promediar el número de etiquetas máximo que puede poseer una instancia.

Precisión multi-etiqueta

El cálculo de la precisión en el caso multi-etiqueta se indica en la Ecuación 2.25.

$$p_{total} = \frac{\sum_{j=1}^{M} p_j}{M} \tag{2.25}$$

donde p_j representa la precisión obtenida del clasificador de la j-esima etiqueta y M es el número de etiquetas que posee cada muestra, que para el caso que nos compete es de cinco etiquetas por pregunta.

Exhaustividad multi-etiqueta

Para calcular la exhaustividad en la clasificación multi-clase se emplea la Ecuación 2.26.

$$r_{total} = \frac{\sum_{j=1}^{M} r_j}{M} \tag{2.26}$$

En la ecuación, r_j es la exhaustividad obtenida del clasificador de la j-esima etiqueta y M es el número de etiquetas que tiene cada muestra.

Valor F1 multi-etiqueta

El valor F1 en clasificación multi-clase se define en la ecuación 2.27.

$$F1_{total} = \frac{\sum_{j=1}^{M} F1_j}{M} \tag{2.27}$$

donde $F1_i$ es el valor F1 calculado de la j-esima etiqueta y M es el número de etiquetas que posee cada muestra.

Como fue posible apreciarse a lo largo de este capítulo, la construcción de un sistema de clasificación es una tarea que debe considerar varios componentes, principalmente cuando los datos a procesar no se encuentran caracterizados previamente.

La selección de un modelo de clasificación es también importante no solo por las implicaciones que pueda tener en cuanto a consumo de recursos computacionales, sino por que la elección entre un modelo u otro abre la posibilidad de encontrarle propiedades a los datos que hayan sido pasadas por alto.

Aunque hubiera sido deseable que no se tuviera que aminorar el efecto de la "maldición de la dimensionalidad", resultó sumamente provechoso el explorar la temática de selección de características, considerando que es un problema que aparece constantemente.

Se espera que con las métricas definidas se pueda apreciar la efectividad de los sistemas de clasificación diseñados y sobre todo sea posible establecerse un marco de comparación común contra otros desarrollos.

A partir de este momento, empleando las técnicas y algoritmos descritos en este capítulo, es posible integrar los sistemas de clasificación que sean capaces de procesar los datos.

Capítulo 3

Diseño e implementación

Haciendo uso de las ideas planteadas en el capítulo anterior, se desarrollaron una serie de tareas que conforman el sistema planteado para esta tesis.

Este capítulo describe el funcionamiento de dos métodos de preprocesamiento para los datos, los cuales tienen como objetivo eliminar elementos innecesarios del texto y mantener aquellas palabras que resulten relevantes para la clasificación y poder generar vectores de características.

Posteriormente se discute el método utilizado de almacenamiento de datos. Este método fue necesario porque se presentaron algunas dificultades a causa del tamaño de los vectores de características y el número de estos.

A su vez, se presentan dos métodos de clasificación multi-clase y multi-etiqueta, uno basado en la distribución de probabilidad de un clasificador Naive Bayes, y un sistema de clasificación por conjunto de clasificadores.

Sin embargo, durante el diseño y prueba de las diferentes tareas planteadas surgieron varios problemas asociados con el uso de la memoria de los sistemas de cómputo empleados, Esta problemática se describe en la Sección 3.5.

Finalmente se muestran las técnicas de selección de características utilizadas por este sistema.

3.1. Preprocesamiento

Como se describió en la Sección 2.1, es conveniente eliminar todos los elementos del texto que se consideren innecesarios para poder realizar una adecuada categorización del mismo. Esto hace necesario incorporar algunas

tareas orientadas a este propósito. Se presentan las implementaciones de estas ideas en esta sección así como una técnica que se presentó al momento de utilizar la implementación del algoritmo Bag of Words de la librería sklearn [Pedregosa et al., 2011].

3.1.1. Preprocesamiento Tradicional

Se identificó en primera instancia que los signos de puntuación no aportan información que pueda utilizar el clasificador y mas bien actúan como ruido, tanto para el proceso de clasificación como para la selección de características. Esto se debe a que, por reglas gramaticales, los signos de puntuación van concatenados a las palabras, lo que la vuelve en una palabra diferente para el algoritmo de caracterización (ejemplo: *entonces* y *entonces*, son dos palabras diferentes para *Bag of Words*). Además, el uso de los signos de puntuación es relativamente errático, ya que en algunos casos se utilizan como herramientas de formato de la pregunta, ya sea para delimitar secciones de ésta (por ejemplo líneas de texto compuestas únicamente por guiones o dos puntos). Se usan también para resaltar secciones de la pregunta como errores en código o sucesos a causa de un programa o segmento de código (por ejemplo un guión seguido de el símbolo mayor que a modo de flecha), resultando en dimensiones adicionales que generalmente afectan sólo a una pregunta.

Además de a los signos de puntuación, se consideró conveniente eliminar todas las palabras inusuales de las preguntas, siendo estas palabras inusuales tales como: nombres de variables, nombres propios, modismos, números, etc. Estas palabras tampoco aportan información relevante al clasificador. Para lograrlo se utilizó la librería NLTK (*Natural Language ToolKit* [Loper and Bird, 2002]), la cual posee un listado de palabras reconocidas en el idioma inglés. Durante el preprocesamiento todas las palabras que componen a cada pregunta se buscan en este listado y si no existen se eliminan. Sin embargo fue evidente que ciertas palabras o siglas aunque no sean comunes en el lenguaje diario, lo son en el contexto en el que fueron escritas las preguntas. Por ejemplo, los acrónimos o nombres de programas, protocolos, lenguajes y demás elementos que componen a las ciencias de computación. Considerando lo anterior, se agregó un léxico con diversas palabras y siglas usadas en temas de computación. De esta forma se pueden eliminar palabras innecesarias o bien con poco aporte para la clasificación, a la vez que se preservan palabras que son altamente informativas para ubicar las preguntas en una clase.

En el Algoritmo 9 se indican los pasos realizados para eliminar las pala-

bras inusuales del texto de las preguntas. Este algoritmo describe una función que recibe como parámetros una cadena de texto y una lista de palabras denominada vocabulario. Se separa el texto en palabras individuales (línea 2) las cuales se comparan con el vocabulario original, y mediante la función `diferencia` (línea 3)se obtienen las palabras que no se encuentran en el vocabulario pero si en la lista de palabras (de manera equivalente en teoría de conjuntos a $A \triangle B$) las cuales se almacenan en la lista `inusuales`. Posteriormente, por cada palabra inusual presente en la lista de inusuales (línea 4) esta se elimina del texto (línea 5). Finalmente se devuelve el texto depurado.

Algoritmo 9: Eliminación de palabras inusuales

```
1 function remueve_inusuales(texto, vocabulario):
      Datos: texto el texto de la pregunta
             vocabulario un diccionario con las palabras usuales para
             las preguntas
      Resultado: El texto de la pregunta con las palabras inusuales
                 removidas
2     palabras ← separa(texto)
3     inusuales ← diferencia(palabras, vocabulario)
4     for inusual in inusuales do
5         texto ← elimina(inusual, texto)
6     end
7     return texto
```

Otro elemento que se consideró innecesario para la clasificación fueron las etiquetas HyperText Markup Language (HTML) que componen a las preguntas. StackOverflow proporcionó las preguntas tal cual fueron escritas usando el editor de texto integrado en el sitio. El editor ofrece herramientas de formateo de texto empleando precisamente etiquetas HTML para este propósito. Para lidiar con las etiquetas HTML se desarrolló el Algoritmo 10. Para excluirlas, se empleó un analizador sintáctico (*parser*) de HTML llamado Beautiful Soup [Richardson, 2004], el cual posee diferentes métodos para la manipulación de documentos HTML. Para eliminar las etiquetas de formato de texto se crea un objeto Beautiful Soup donde se le indica que la pregunta es el documento HTML que debe interpretar (línea 2). Luego, se invoca el método `obten_texto()` del objeto Beautiful Soup, este método devuelve el documento HTML con el que se creó el objeto pero sólo muestra

el texto que lo componen y no las etiquetas que le dan formato (línea 3). El texto sin etiquetas HTML se encuentra en codificación unicode, por lo que se pasa por un filtro para dejarlo en código ASCII (línea 4).

Algoritmo 10: Eliminación de etiquetas HTML

```
1 function remueve_html(html):
      Datos: La pregunta en formato HTML
      Resultado: El texto de la pregunta con las etiquetas HTML
                 removidas y en caracteres imprimibles
2     parser ← BeautifulSoup(html)
3     texto_unicode ← parser.obten_texto()
4     texto ← filtra_unicode(texto_unicode)
5     return texto
```

Se incorporó un *stemmer* al preprocesamiento para reducir el número de características innecesarias. Un stemmer identifica la raíz de una palabra y elimina la parte de conjugación o declinación de esta. Por ejemplo observó, observaron y observamos son conjugaciones del verbo `observar`, las cuales el Stemmer reduce a `observ` ya que esa es la raíz de cada una de estas palabras. La transformación que se realizó a las preguntas se enfoca en las palabras presentes en la pregunta y no al contexto que tienen entre ellas, por tanto se consideró innecesario tener muchas características que se basan en una sola palabra cuando son variantes de ésta que solo sirven para añadir contexto. El algoritmo 11 indica los pasos descritos, donde, cada palabra presente en el texto es procesada por el stemmer, el cual devuelve la raíz de la palabra (línea 4).

La integración completa de los pasos de eliminación de elementos se puede apreciar en el Algoritmo 12.

Del Algoritmo 12 se observa que, a cada pregunta, primero se le eliminan las etiquetas HTML (línea 2). Lo mismo se hace con los signos de puntuación (líneas 3 y 4). Se convierte el texto a minúsculas (línea 6). Se eliminan las palabras inusuales (línea 7). Se eliminan las palabras no significativas (líneas 8, 9 y 10). Finalmente, se identifican las palabras raíz (línea 12) y se devuelve el texto depurado de los elementos innecesarios.

Como se puede observar, este procedimiento se debe repetir por cada pregunta presente en el conjunto de datos. El Algoritmo 13 muestra cómo se realiza la iteración de las preguntas. Por cada pregunta contenida en el archi-

Algoritmo 11: Eliminación de lemas

```
1  function remueve_lemas(texto, stemmer):
       Datos: texto el texto de la pregunta
              stemmer el stemmer que obtiene las raíces de las palabras
       Resultado: El texto de la pregunta con las palabras sin lemas
2      parcial ← separa(texto)
3      for palabra in parcial do
4          texto_nuevo ← stemmer.stem(palabra)
5      end
6      return texto_nuevo
```

Algoritmo 12: Preprocesamiento por pregunta

```
1  function preprocesa_pregunta(pregunta, puntuación, vocabulario,
   stemmer, stopwords):
       Datos: pregunta la pregunta en formato HTML
              puntuacion un arreglo que contiene los signos de
              puntuación
              vocabulario el vocabulario de palabras usuales para las
              preguntas
              stemmer el stemmer para obtener las raíces de las palabras
              stopwords una lista que contiene las palabras no
              significativas (stopwords)
       Resultado: Un arreglo con las palabras que componen a la
                  pregunta
2      texto ← remueve_html(pregunta)
3      for signo in puntuacion do
4          texto ← texto.reemplaza(signo, ")
5      end
6      texto ← minúsculas(texto)
7      texto ← remueve_inusuales(texto, vocabulario)
8      inutiles ← diferencia(palabras, stopwords)
9      for palabra in inutiles do
10         texto ← elimina(palabra, texto)
11     end
12     texto ← remueve_lemas(texto, stemmer)
13     return texto
```

vo CSV (línea 8), estas se van añadiendo a una lista llamada `preguntas`, la cual se va formando con el resultado del Algoritmo 12. Nótese que cada 1000 preguntas procesadas estas se almacenan en el disco duro para evitar saturar la memoria (líneas 11 y 12) para después reinicializar el arreglo `preguntas` (línea 13). Antes de que finalice el procedimiento se almacenan las preguntas restantes en el disco duro (línea 15). La notación *nltk* indica objetos y métodos invocados de la librería NLTK.

Algoritmo 13: Preprocesamiento general

```
1  function preprocesamiento(archivo_csv):
       Datos: archivo_csv el archivo csv que contiene las preguntas
       Resultado: Las preguntas transformadas para ser vectorizadas
2      vocabulario ← carga_vocabulario(ruta_vocabulario)
3      puntuacion ← puntuaction()
4      stemmer ← nltk.PorterStemmer()
5      stopwords ← nltk.Stopwords()
6      n ← 0
7      preguntas ← [ ]
8      for pregunta in archivo_csv do
9          añade(preguntas, preprocesa_pregunta(pregunta, puntuacion,
           vocabulario, stemmer, stopwords))
10         n ← n + 1
11         if n mod 1000 = 0 then
12             baja_disco(preguntas)
13             preguntas ← []
14     end
15     baja_disco(preguntas)
```

Para lograr que la caracterización fuera sobre el modelo BOW se utilizó un objeto `CountVectorizer` de la librería sklearn [Pedregosa et al., 2011], el cual aplica Bag of Words a los datos que se le proporcionen. En el Algoritmo 14 es posible ver que el objeto vectorizador recibe un archivo parcial (línea 3), el cual es el que genera y puebla el Algoritmo 13 al invocar la función `baja_disco`. La notación *sklearn* indica métodos y objetos invocados de la librería sklearn.

Aunque las técnicas mencionadas fueron capaces de eliminar en gran medida las palabras no significativas y las palabras inusuales de las preguntas,

Algoritmo 14: Vectorización de las preguntas

```
1 function vectorizacion(archivo_parcial):
      Datos: archivo_parcial el archivo donde se almacenaron las
             preguntas durante el preprocesamiento
      Resultado: La matriz que contiene las preguntas una vez
                 vectorizadas
2     vectorizador ← sklearn.CountVectorizer()
3     vectores ← vectorizador.trans(archivo_parcial)
4     return vectores
```

en algunas de estas sucedió que parte de código HTML no se encontraba estructurado de manera adecuada. Pese a que las etiquetas HTML fueron detectadas y eliminadas al ser palabras inusuales, no fue así con ciertas palabras o frases contenidas dentro de las etiquetas HTML. Lo anterior ocasionó que quedaran palabras o frases sin sentido, haciendo que el número de características detectadas por el el algoritmo BoW fuera sumamente alto (generando vectores con poco mas de dos millones de características), y por consiguiente el manejo de los datos fuera sumamente complicado. Sin embargo, este problema se resolvió fácilmente indicándole al vectorizador que ignorara todas las palabras que no aparezcan en al menos el 1 % de los datos, logrando reducir el número de características a 665. Se tomó esta decisión por que aunque estas palabras eran potencialmente benéficas para identificar con alta precisión a ciertas preguntas, el costo computacional de mantenerlas (tanto para el almacenamiento como para su uso posterior en los procesos de entrenamiento y predicción) era muy alto.

3.1.2. Preprocesamiento a través del Vectorizador

Si bien la manipulación de uno de los parámetros del vectorizador logró reducir considerablemente el número de características surgió una inquietud: ¿Qué pasaría si en vez de hacer un paso de preprocesamiento se manda de manera directa los datos al vectorizador y se le indican umbrales para la selección de palabras a tomar como características y que se deshaga tanto de las palabras no significativas como de las palabras inusuales?

Para hacerlo únicamente se define una clase intermedia que va a contener el archivo CSV (*Comma-Separated Values*) donde se encuentran las pregun-

tas con el fin de recuperar solo el título y cuerpo de la pregunta. Esta nueva cadena se almacena en un objeto iterable temporal el cual procesa el vectorizador para realizar la transformación de los datos. Al vectorizador se le modificaron dos parámetros: `min_df` el cual indica la frecuencia documental (esto es, el número de veces que aparece una palabra en un conjunto de documentos) mínima que debe poseer para ser considerada como característica; `max_df` indica por el contrario la frecuencia documental máxima que puede tener una palabra para no ser excluida de las características. De esta manera se genera un preprocesamiento *in situ* que forma un vocabulario particular para el conjunto de datos dado. Aunque no se realizó en esta tesis, es posible recuperar el vocabulario formado por el objeto vectorizador accediendo a este miembro del objeto de manera directa.

El Algoritmo 16 se encarga de caracterizar la salida del método `lector_flojo`, descrito en el Algoritmo 15. El Algoritmo 15 acumula la concatenación del título y contenido tal y como fueron escritos de cada pregunta en una lista (línea 4), una vez que alcanza el final de archivo retorna esta lista.

Algoritmo 15: Lectura del archivo de entrenamiento a través de un lector flojo

```
1 function lector_flojo(archivo_original):
      Datos: archivo_original el archivo csv con los datos de
             entrenamiento
      Resultado: Un acumulador del texto seleccionado
2     lector_csv ← lee_csv(archivo_original)
3     while lector_csv.siguiente do
4         acumula(lector_csv[1] + ' ' + lector_csv[2])
5     end
```

De manera semejante al Algoritmo 14, el Algoritmo 16 genera un objeto `CountVectorizer` pero en vez de procesar el archivo parcial, procesa el resultado del Algoritmo 15, y empleando los parámetros `min_df` y `max_df` se discriminan los elementos infrecuentes y los mas frecuentes (línea 3).

3.2. Almacenamiento de datos

Ya que estuvo definido el preprocesamiento de las preguntas se procedió a caracterizarlas y almacenarlas como vectores de características, para después

Algoritmo 16: Vectorización por preprocesamiento simple

```
1 function vectorizacion(archivo_original):
      Datos: archivo_original el archivo csv con los datos de
             entrenamiento
      Resultado: La matriz que contiene las preguntas una vez
                 vectorizadas
2     vectorizador ← sklearn.CountVectorizer(min_df=0.01,
      max_df=0.85)
3     vectores ← vectorizador.trans(lector_flojo(archivo_original))
4     return vectores
```

ser usadas en las diferentes metodologías de clasificación que se diseñaron para atacar el problema planteado en este trabajo. Sin embargo algo que pareciera ser trivial resultó ser de las primeras dificultades vistas en este desarrollo.

Los datos que se decidieron almacenar son: la matriz de características que representan a las preguntas; una lista de tuplas con el nombre de la etiqueta y un entero para identificar las preguntas. Se asigna un identificador numérico a cada etiqueta y una matriz que contiene los identificadores de todas las etiquetas asignadas a cada pregunta (la cual es de dimensiones $5 \times N$ donde N es el total de preguntas almacenadas). Debido a que no todas las preguntas poseen cinco etiquetas, si la pregunta tiene menos de cinco etiquetas se asigna un -1 para ese valor de etiqueta (se reservó el 0 como identificador de etiqueta).

En una primera aproximación se empleó el formato utilizado de manera nativa por la librería Numeric Python (Numpy [van der Walt et al., 2011]) llamado NPY. Este formato almacena la información de estructura y tipo de dato necesarios para reconstruir los arreglos almacenados independientemente de la arquitectura de la computadora.

Empleando este esquema se pudieron almacenar las matrices resultantes del proceso de preprocesamiento y vectorización, pero surgieron varias dificultades con su uso, particularmente con el manejo de la memoria del sistema.

Se requirió de un formato de almacenamiento mas robusto que pudiera permitir la recuperación de los datos de manera parcial. Esta carga parcial puede suceder ya sea contando cierto número de filas que componen las ma-

trices, o bien recuperar vectores de manera condicional (por ejemplo: leer únicamente los vectores que contienen una etiqueta en particular). Además, es deseable que se tenga la posibilidad de almacenar de manera separada cada una de las matrices sin que esto suponga tener que leer en cierto orden los datos.

Se optó por utilizar PyTables [Alted et al., 2002], el cual es un paquete de administración jerárquica de conjuntos de datos orientado al manejo de grandes cantidades de datos. PyTables permitió almacenar los datos organizados en diferentes tablas, las cuales pueden accederse de manera independiente. A su vez estas tablas pueden ser iteradas tanto de manera secuencial como empleando una lista, lo cual permite leer partes de los datos para procesarlos y realizar operaciones con ellos sin comprometer grandes cantidades de memoria. De modo semejante a una base de datos relacional se pueden realizar consultas sobre los datos lo cual permite recuperar vectores de manera condicional, una tarea que se requirió para uno de los enfoques de clasificación que no hubiera sido posible realizar (dada la cantidad de memoria disponible en los sistemas utilizados) empleando formatos de archivo menos sofisticados.

De esta manera fue posible almacenar la información caracterizada haciendo posible la manipulación de estos sin comprometer en exceso la memoria, a la vez de que permite realizar operaciones que no serían posibles (dado el marco de tiempo limitado) si se empleara acceso secuencial a los datos.

3.3. Técnicas de clasificación

Debido a la enorme cantidad de etiquetas fue necesario aplicar ciertas restricciones al proceso de clasificación. Se identificaron las 500 etiquetas mas frecuentes y se seleccionaron las preguntas que poseen estas etiquetas. Se eligió este número de etiquetas por ser el empleado en [Parikh, 2013], ya que como se menciona en ese artículo, las 500 etiquetas mas frecuentes representan alrededor del 60 % del total de etiquetas usadas. El resto de las etiquetas no se consideró para la clasificación debido a que el soporte individual de estas etiquetas (el número de preguntas que poseen estas etiquetas) es muy bajo.

Como se explica a continuación se definieron dos enfoques de clasificación. Ambos solamente se entrenaron con las preguntas que tienen estas etiquetas mas frecuentes, el resto de preguntas se dejó intacto. Tanto el conjunto de entrenamiento como el de prueba fueron conformados por preguntas or-

denadas aleatoriamente que poseen tanto las etiquetas frecuentes como las no frecuentes, pero al momento de entrenar un clasificador se verifica en el conjunto de entrenamiento que la pregunta posea una de las etiquetas mas frecuentes, de no ser así no se utiliza como muestra de entrenamiento. Esta validación no sucede con el conjunto de prueba.

Pareciera escaso identificar tan solo las 500 etiquetas mas frecuentes, pero estas 500 etiquetas conforman una mayoría que se consideró suficiente para obtener resultados de clasificación favorables.

El proceso de clasificación se realiza de la siguiente manera: del subconjunto de los datos se toma un 90 % para que funcione como conjunto de entrenamiento y un 10 % como conjunto de prueba. Como se ha mencionado anteriormente, se intenta mantener la menor cantidad de memoria comprometida, para lo cual se va entrenando al clasificador con un valor arbitrario de preguntas, el cual corresponde a una fracción del total del conjunto de entrenamiento y debe ser suficientemente grande para maximizar el uso de la memoria. Este valor debe permitir particionar al conjunto de entrenamiento en segmentos del mismo tamaño.

Se definieron dos enfoques para realizar el proceso de clasificación los cuales se describen brevemente a continuación.

3.3.1. Sistema Basado en Distribución de Probabilidad

El primer enfoque fue el utilizar la distribución de probabilidad generada por un clasificador probabilista para resolver el problema de clasificación multi-etiqueta. Naive Bayes al ser un clasificador probabilista, genera una distribución de probabilidad de que una muestra pertenezca a cada una de las clases que conoce el clasificador.

Se utilizó el clasificador Naive Bayes implementado en la librería sklearn debido a que resultó fácil de integrar al esquema de desarrollo diseñado y además posee una serie de funciones adicionales útiles para el propósito de este trabajo. En los algoritmos la notación *sklearn* indica que es un objeto o método invocado de la librería sklearn.

Para entrenar el clasificador primero se preparan los datos a utilizar para entrenarlo (Algoritmo 17). Se leen de disco el conjunto de entrenamiento `X`, su matriz de etiquetas `y` y una lista de etiquetas (línea 2). En la lista `etiquetas` se encuentran las 500 etiquetas mas frecuentes, por lo que por cada pregunta se comprueba que se encuentre en esta lista (líneas 5 y 6). Se usa la primera etiqueta asignada a la pregunta como la clase que distingue

a la pregunta (línea 6); se asume que el autor de la pregunta consideró a la primera etiqueta como la mas significativa. En los arreglos `indices` y `y_n` se almacenan los índices y las etiquetas a emplear para el entrenamiento (líneas 7 y 8). Estos arreglos se mezclan aleatoriamente de manera simultanea para no perder su relación entre si (línea 11). Luego, con los índices y etiquetas seleccionados, se invoca la función `entrena` la cual realiza las operaciones de entrenamiento (línea 12).

Algoritmo 17: Preparación de datos para clasificar mediante SBDP

```
1  function clasificación_sbdp(ruta_archivo):
       Datos: ruta_archivo la ruta del archivo donde fueron almacenados
              los vectores de prueba y sus respectivas etiquetas
       Resultado: El clasificador entrenado
2      X_ent, y, etiquetas ← lee_archivos(ruta_archivo)
3      índices ← [ ]
4      y_n ← [ ]
5      for fila in y do
6          if fila[0] in etiquetas then
7              agrega(índices, fila.index)
8              agrega(y_n, fila[0])
9          end
10     end
11     índices, y_n ← sklearn.mezcla(índices, y_n)
12     clf ← entrena(índices, X_ent, y_n, 1)
13     return clf
```

Como se aprecia en el Algoritmo 18, el clasificador se va entrenando empleando una técnica donde el entrenamiento sucede por lotes de muestras (líneas 13, 14 y 15), lo cual se hace separando en segmentos la matriz `X` y el vector de etiquetas `y`; en condiciones ideales es preferible usar el total de muestras, pero debido al número de muestras a procesar no es posible para este caso. Estas operaciones se realizan hasta que se procesan todas las preguntas del conjunto de entrenamiento. Para lograrlo se emplea la función `partial_fit` que posee los clasificadores Naive Bayes y SGDC (en inglés *Stochastic Gradient Descent Classifier* o bien Clasificador Descenso por Gradiente Estocástico) implementados en sklearn, la cual permite que el entrenamiento suceda en lotes de muestras (línea 15). Nótese que se puede especificar

el tipo de clasificador a entrenar (líneas 4 y 7), esto con la intención de reusar esta función para el segundo esquema de clasificación propuesto.

Algoritmo 18: Entrenamiento del clasificador SBDP

```
1  function entrena(índices, X_ent, y, tipo_clasificador):
       Datos: indices los índices de las preguntas a ser procesadas
              X los vectores que representan a las preguntas
              y el vector de etiquetas
              tipo_clasificador un entero que indica que tipo de
              clasificador entrenar
       Resultado: el clasificador clf
2      inicio ← 0
3      clf ← Nulo
4      if tipo_clasificador = 1 then
5          clf ← sklearn.NaiveBayes()
6      end
7      else
8          clf ← sklearn.SGDClassifier()
9      end
10     cuantas ← ⌊ longitud(índices) / 1000 ⌋
11     for i in rango(cuantas) do
12         fin ← (i + 1) * longitud(índices) / cuantas
13         y_entrenamiento ← separa(y, inicio, fin)
14         x_entrenamiento ← X_ent[índices[inicio:fin]]
15         clf.partial_fit(x_entrenamiento, y_entrenamiento)
16         inicio ← fin
17     end
18     x_entrenamiento ← X[índices[fin:]]
19     clf.partial_fit(x_entrenamiento, y_entrenamiento)
20     return clf
```

En el Algoritmo 19, para realizar las predicciones se usa la función `predict_proba()` proporcionada por la implementación de sklearn del clasificador (línea 6), la cual devuelve la distribución de probabilidad que tiene una pregunta de pertenecer a cada una de las etiquetas que pudo observar el clasificador durante el entrenamiento. Esta distribución está dada como un arreglo donde la posición en el arreglo indica la clase y el valor corresponde a

la probabilidad de pertenecer a dicha clase. Para seleccionar las clases se forman tuplas con el identificador de la clase y la probabilidad que le asignó el clasificador a la muestra de pertenecer a esa clase (línea 7); se ordenan las probabilidades de manera descendente y se guardan las primeras cinco predicciones (líneas 8 y 9). Si alguna de estas cinco probabilidades fuera cero, se reemplaza el identificador de la etiqueta por el identificador de la etiqueta vacía (líneas 11-18). Una vez obtenidas estas predicciones se comparan con las etiquetas reales y se determinan las puntuaciones del clasificador.

3.3.2. Sistema Clasificador por Conjunto de Clasificadores

En este enfoque se entrenan cinco clasificadores no binarios donde cada clasificador se encarga de predecir cada etiqueta de una pregunta.

A diferencia del enfoque anterior, no se requiere recuperar la distribución de probabilidad, por lo que cada clasificador predice de manera directa la etiqueta que considera le corresponde a la pregunta. En este caso fue necesario que la "etiqueta vacía" fuera parte de las etiquetas conocidas por los clasificadores que predicen de la segunda a la quinta etiqueta (no hay preguntas sin ninguna etiqueta).

Para entrenar a los clasificadores primero se determina qué posición de etiqueta se va a procesar (recordando que cada pregunta posee de una a cinco etiquetas). Posteriormente, utilizando la lista de las 500 etiquetas mas frecuentes, se seleccionan las preguntas que posean estas etiquetas en la posición seleccionada. Con este subconjunto de preguntas se entrena al clasificador para la posición seleccionada. Si una pregunta no posee etiqueta alguna para esa posición, se le asigna la etiqueta `etiqueta-vacia`.

Para realizar la predicción las preguntas se pasan por los cinco clasificadores los cuales indican su predicción para las cinco etiquetas que tiene asignada una pregunta, obviamente considerando la ausencia de etiqueta como una etiqueta extra.

En el caso de que haya predicciones repetidas, producidas por clasificadores diferentes para etiquetas diferentes, estas se eliminan y se sustituyen por la etiqueta vacía.

El funcionamiento del Algoritmo 20 es idéntico al enfoque anterior (Algoritmo 17), salvo que la invocación de `clasificacion_conjunto` incluye el número de etiqueta a predecir (de 1-5), y al momento de seleccionar la

Algoritmo 19: Predicción a través de SBDP

```
1  function predice_sbdp(índices_prueba, clf, X, y):
       Datos: indices_prueba los índices de los vectores con los que se va
              a probar el clasificador
              clf el clasificador entrenado
              X la matriz que contiene los vectores
              y el vector de etiquetas
       Resultado: Las predicciones generadas por el clasificador y las
                  etiquetas reales
2      y_pred ← [ ]
3      X_prueba ← selecciona(X, índices_prueba)
4      y_real ← selecciona(y, índices_prueba)
5      for x in X_prueba do
6          prediccion ← clf.predict_proba(x)
7          a ← [(clf.classes_[prediccion.index(y)], probabilidad) for
           probabilidad in prediccion]
8          a ← ordena (a, key=lambda prob: prob[1], reverse=True)
9          a ← recorta(a, 5)
10         b ← [ ]
11         for p in a do
12             if p[1] > 0 then
13                 b = agrega(b, p[0])
14             end
15             else
16                 b = agrega(b, etiqueta-vacia)
17             end
18             y_pred ← agrega(predicciones, b)
19         end
20     end
21     return y_pred, y_real
```

etiqueta que representa a la pregunta se utiliza la variable `etiqueta` (línea 9).

Algoritmo 20: Preparación de datos para clasificar mediante SCC

```
1  function clasificacion_scc(ruta_archivo, etiqueta):
       Datos: ruta_archivo la ruta del archivo donde fueron almacenados
                  los vectores y sus respectivas etiquetas
              etiqueta un entero que indica la etiqueta a predecir
       Resultado: El clasificador entrenado para la etiqueta indicada
2      X_ent, y, etiquetas ← lee_archivos(ruta_archivo)
3      índices ← [ ]
4      y_n ← [ ]
5      if etiqueta > 1 then
6          agrega(etiquetas, etiqueta-vacia)
7      end
8      for fila in y do
9          if fila[etiqueta] in etiquetas then
10             agrega(índices, fila.index)
11             agrega(y_n, fila[etiqueta])
12         end
13     end
14     índices, y_n ← sklearn.mezcla(índices, y_n)
15     clf ← entrena(índices, X_ent, y_n)
16     return clf
```

La predicción por conjunto de clasificadores se realiza recuperando la predicción de cada clasificador. El Algoritmo 21 muestra como se realizan estas predicciones. Se itera seleccionando un sub-conjunto del conjunto de prueba (línea 13); esto se realiza nuevamente por restricciones de memoria. Este subconjunto se pasa a los clasificadores, cuyas predicciones se almacenan en el arreglo `todos` utilizando la función `apila_columna` la cual va concatenando los datos no por fila (como se interpreta tradicionalmente una operación de arreglos) sino por columna, esto con el propósito de generar una matriz de predicciones semejante a la matriz de etiquetas original (línea 15). Se comprueban las predicciones individuales, si alguna predicción tuviera repetidos (es decir, que dos o mas clasificadores hayan dado el mismo resultado) estas repeticiones se eliminan y se reordenan las predicciones para colocar en las

etiquetas finales la etiqueta vacía (línea 16), esto debido a que es altamente probable que una pregunta posea tres o menos etiquetas (no confundir el número de clases conocidas con el número de etiquetas que puede poseer individualmente cada pregunta). las predicciones se insertan en el arreglo `y_pred` esta vez for fila y preservando la posición de las predicciones (líneas 17 y 18). El Algoritmo 22 enseña el procedimiento de eliminación de etiquetas repetidas.

El Algoritmo 22 realiza una operación de filtrado de predicciones donde, escrito de manera muy compacta, genera un arreglo `r` con tantos elementos `x` se encuentren contenidos en `z` si no se encuentran en el conjunto `vistos` o si no se han añadido a `vistos` (línea 4), lo que hace es que si no lo ha incluido al conjunto `vistos` lo agrega a `r` pero también al conjunto `vistos` usando la función `suma_vistos`, así si un elemento `x` llegara a ser igual a otro anterior ya no lo agrega a `r`. De esta manera se preserva la posición de la predicción (que es crucial en este enfoque) y toda predicción repetida se remueve. Posteriormente se comprueba que el arreglo `r` tenga cinco elementos, de no ser así se le agregan cuantos elementos sean necesarios todos con la etiqueta vacía (líneas 5 y 6).

3.4. Selección de características

De acuerdo a lo descrito en la Sección 2.3, se utilizaron dos mecanismos de selección de características: Umbral de Varianza y SVC lineal. La implementación de estas técnicas viene incluida en la librería sklearn, por lo que su integración a esta tesis fue realizada sin mayor inconveniente.

La técnica umbral de varianza fue invocada empleando:

```
vt = VarianceThreshold(threshold=0.16)
```

El valor de $threshold = 0.16$ fue seleccionado debido a que se encontró adecuado para eliminar el número suficiente de características con poca varianza sin comprometer la separación espacial de los vectores de características.

La implementación de LinearSVC se realizó empleando:

```
lsvc = LinearSVC(C=0.0005, penalty="l1", dual=False)
```

Como se menciona en la Sección 2.3.2, el uso de esta técnica requiere que se seleccione una función de penalización (en este caso se usa la norma L_1) y un parámetro C (que representa la constante de capacidad usada en 2.11) que

Algoritmo 21: Predicción a través de SCC

```
1  function prediccion_scc(índices_prueba, clfs, X, y, umbral):
       Datos: indices_prueba los índices de prueba
              clfs una lista que contiene los clasificadores entrenados
              X los vectores que representan a las preguntas
              y el conjunto de etiquetas y
              umbral un número entero que indica cuantas preguntas
              procesar por ciclo
       Resultado: Las predicciones y las etiquetas reales separadas por
                  posición
2      y_pred ← [ ]
3      y_real ← [ ]
4      x_prueba ← X[índices_prueba]
5      for fila in y do
6          for i in range(5) do
7              agrega(y_real[i], fila[i])
8          end
9      end
10     inicio ← 0
11     fin ← 0
12     cuantas ← longitud(x_prueba) / umbral
13     for i in range(cuantas) do
14         fin ← (i + 1) * longitud(x_prueba) / cuantas
15         todos = apila_columna ((clfs[0].predict(x_prueba[inicio:fin]),
           clfs[1].predict(x_prueba[inicio:fin]),
           clfs[2].predict(x_prueba[inicio:fin]),
           clfs[3].predict(x_prueba[inicio:fin]),
           clfs[4].predict(x_prueba[inicio:fin])))
16         z = [filtra(x) for x in todos]
17         for j in range(5) do
18             apila_fila(y_pred[j], [x[j] for x in z])
19         end
20         inicio ← fin
21     end
22     todos = apila_columna ((clfs[0].predict(x_prueba[inicio:]),
       clfs[1].predict(x_prueba[inicio:]), clfs[2].predict(x_prueba[inicio:]),
       clfs[3].predict(x_prueba[inicio:]), clfs[4].predict(x_prueba[inicio:])))
23     z = [filtra(x) for x in todos]
24     for j in range(5) do
25         apila_fila(y_pred[j], [x[j] for x in z])
26     end
27     return y_pred, y_real
```

Algoritmo 22: Filtrado de predicciones repetidas

```
1 function filtra(z):
      Datos: z Una lista con datos repetidos
      Resultado: La lista z sin los datos repetidos
2     vistos ← conjunto(vacio)
      /* se instancia la función suma del conjunto vistos
         para usarla independientemente                    */
3     suma_vistos = vistos.suma
4     r = [x for x in z if not (x in vistos or suma_vistos(x))]
5     for i in range(5) - longitud(r) do
6         agrega(r, etiqueta-vacia)
7     end
8     return r
```

sirve para controlar el impacto del error calculado por la SVC (comentada en la Sección 2.2.2). Se usó $C = 0.0005$ ya que ocasiona una reducción de un 50-70 % del número de características totales (este valor se seleccionó empíricamente ejecutando repetidamente este proceso con diferentes valores) y el castigo impuesto L_1 ocasiona que se tengan soluciones dispersas [Zhu et al., 2004], y utilizando el método `transform()` incluido en el objeto `LinearSVC` esto ocasiona que seleccione los coeficientes diferentes de cero.

Cómo se discutió anteriormente, umbral de varianza no requiere observar la etiqueta asignada a cada una de las preguntas, por lo que es posible utilizarlo de manera directa a las preguntas vectorizadas y generar un conjunto de datos reducido. En cambio, Linear SVC requiere observar la etiqueta asignada a la pregunta debido a que en esencia es un procedimiento de clasificación, por lo que fue necesario adaptar el procedimiento de entrenamiento para que se seleccionen las características por etiqueta asignada y así optimizar la selección de características lo cual ocasiona que para usar este método el proceso de selección de características debe ser repetido por cada una de las etiquetas a predecir.

3.5. Estrategias de manejo de memoria

Durante el desarrollo de las funciones que componen esta tesis aparecieron una serie de problemas provenientes de la administración de la memoria, los

cuales pudieron ser solucionados utilizando alternativas de estructuras de datos y algoritmos.

Un caso de estas alternativas es posible verse en el Algoritmo 13, donde en vez de dejar que la lista de preguntas procesadas siga recibiendo nuevas preguntas de manera indefinida, se hace uso de un archivo temporal para posteriormente limpiar la lista y recuperar memoria ocupada.

Ideas semejantes se pueden observar en el Algoritmo 18, donde aunque el tamaño en memoria de la matriz de características se vio reducido, al momento de invocar el clasificador para entrenarlo fue imposible hacerlo con el total de los datos por lo que fue necesario emplear el método `partial_fit` que poseen los clasificadores Naive Bayes y SGDClassifier implementados en sklearn y recorrer la matriz por secciones.

A su vez, algunas operaciones del entrenamiento y predicción se realizan indirectamente utilizando los índices en lugar de los vectores, esto para mantener al mínimo el consumo de memoria de los programas.

La librería sklearn almacena los vectores en una matriz CSC (en inglés *Compressed Sparse Column* o bien Columna Dispersa Comprimida) [Duff et al., 1989]. Las matrices CSC almacenan los valores diferentes de cero que posea una matriz empleando tres arreglos de una dimensión (`val`, `row_ind`, `col_ptr`); en `val` se almacenan los valores diferentes de cero leídos por columna, en `row_ind` almacena los índices por fila de los elementos en el arreglo `val`, esto es, si $\text{val}(k) = a_{i,j}$ entonces $\text{row\_ind}(k) = i$. El arreglo `col_ptr` almacena las ubicaciones en el arreglo `val` que comienzan una columna, esto es, si $val(k) = a_{i,j}$ entonces $col\_ptr(j) \leq k < col\_ptr(j+1)$. De esta manera en lugar de almacenar $n \times m$ valores se almacenan $2nnz + n + 1$ valores, donde nnz es el número de valores diferentes de cero que posea la matriz. Se pudo emplear una matriz CSR (*Compressed Sparse Row* o bien Fila Dispersa Comprimida), pero se prefirió utilizar una matriz CSC esperando agilizar las operaciones de reducción y selección de características mas que las operaciones de iteración de las filas, además la diferencia de uso de memoria entre un tipo de matriz y la otra es inexistente.

Aunque la estructura de datos empleada es altamente eficiente, de manera inicial consumía una cantidad muy grande de memoria, debido al tipo de datos empleado para los arreglos (enteros de 64 bits) por lo que se cambió a enteros de 8 bits sin signo, logrando una reducción significativa del tamaño de la matriz de 3Gb a 650Mb.

Con las funciones implementadas fue posible procesar las preguntas y convertirlas vectores de características, los cuales pudieron ser procesados por

los sistemas de clasificación propuestos. Pese a ciertas dificultades, las operaciones propuestas pudieron realizarse exitosamente en tiempos adecuados y empleando el menor uso de recursos computacionales, y como podrá observarse en el siguiente capítulo, se logró obtener una serie de resultados de acuerdo a los objetivos planteados.

Capítulo 4

Experimentos y resultados

Con los métodos definidos e implementados fue posible realizar una serie de experimentos con el propósito de corroborar la efectividad de estos procesos para cumplir con los objetivos establecidos.

Empleando las métricas descritas en la Sección 2.4 se procedió a medir la efectividad de los sistemas de clasificación desarrollados. En este Capítulo se muestran los promedios de la precisión, exhaustividad y valor F1 obtenidos por cada uno de estos y calculados para el modelo de clasificación multietiqueta.

4.1. Parametros de ejecución de las pruebas

Para la ejecución de las pruebas se empleó el *cluster* de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. Este cluster consta de tres nodos; dos de estos nodos poseen 24 procesadores y 40Gb de memoria y el nodo restante cuenta con 48 procesadores y 70Gb de memoria.

Aunque el desarrollo generado no hace uso de mecanismo alguno de paralelización o cómputo distribuido, se optó por ejecutarlo en el cluster para poder realizar de manera simultánea el entrenamiento y predicción de varios clasificadores.

Como se explicó en la Sección 3.3 se definieron dos sistemas de clasificación: uno empleando la distribución de probabilidad generada por un clasificador Naive Bayes y otro haciendo uso de un conjunto de clasificadores que clasifican de acuerdo a las etiquetas asignadas. El segundo mecanismo hace uso tanto de clasificadores Naive Bayes como de SVM.

Para reducir el tiempo utilizado por estos sistemas durante el entrenamiento y predicción, y además comparar el comportamiento de los clasificadores generados, se optó por utilizar únicamente las 100, 200 y 500 etiquetas mas frecuentes (mas la etiqueta vacía para los clasificadores de la segunda a la quinta etiqueta).

El entrenamiento de los clasificadores se realizó empleando los tamaños de conjunto de entrenamiento indicados en la Tabla 4.1 y empleando 365,590 preguntas como conjunto de prueba. Estos tamaños de entrenamiento y prueba corresponden aproximadamente al 90 % y 10 % del total de datos respectivamente. La selección de las preguntas para conformar los conjuntos de entrenamiento y prueba se realizó de manera aleatoria.

Tabla 4.1: Tamaño de conjuntos de entrenamiento por etiqueta

clases	Primera Etiqueta	Segunda Etiqueta	Tercera Etiqueta	Cuarta Etiqueta	Quinta Etiqueta
100	2758527	1721079	1843326	2633436	3284595
200	3013218	2101518	2042352	2692251	3294720
500	3290310	2600415	2414520	2838330	3326076

El tamaño de los conjuntos de entrenamiento varía debido a que, como se comentó en el capitulo anterior, se utilizaron para entrenar a los clasificadores un sub-conjunto de preguntas que poseen en sus etiquetas las etiquetas mas frecuentes (en la Tabla 4.1 vemos que en la columna clases tenemos 100, 200 y 500 clases), lo cual ocasiona que algunas preguntas no se incluyan debido a que no poseen alguna de estas etiquetas en la posición indicada. Nótese que por ejemplo, la columna Quinta Etiqueta es la mas numerosa para los tres sub-conjuntos, esto debido a que la etiqueta vacía es la mas frecuente para esta posición, haciendo que se incluyan la mayor cantidad de preguntas. De igual manera se puede observar que la columna Primera Etiqueta es la segunda mas numerosa, esto debido a que las etiquetas mas frecuentes por lo general son las que se ponen de manera inicial.

Un ejemplo de predicción sería el arreglo `[4337, 22, 39386, -1, -1]`. Cada valor en el arreglo corresponde al identificador de la etiqueta, los cuales se determinaron en un proceso previo. El identificador 4337 corresponde a la etiqueta `c#`, el 22 a la etiqueta `.net`, el identificador 39386 corresponde a la etiqueta `visual-studio`, y el valor -1 corresponde a la etiqueta vacía.

A continuación se muestran las tablas y figuras ilustrando los resultados de los mecanismos de clasificación descritos en la sección anterior. Las tablas poseen como encabezados los acrónimos de los métodos de preprocesamiento aplicados a los datos. Cada fila representa el resultado obtenido por el sistema

con 100, 200 y 500 clases de entrenamiento. Cada tabla indica los valores obtenidos de la precisión, exhaustividad y valor F1. En todas las tablas se resalta el mejor resultado en negritas.

4.2. Resultados del Sistema Basado en Distribución de Probabilidad

En esta sección se muestran los resultados obtenidos por el sistema basado en la distribución de probabilidad generada por un clasificador Naive Bayes. Los acrónimos utilizados son: Sistema Basado en Distribución de Probabilidad (SBDP), Preprocesamiento Tradicional (PT) que es el preprocesamiento descrito en la Sección 3.1.1, Preprocesamiento Simple (PS) que es el preprocesamiento mostrado en la Sección 3.1.2, Variance Threshold (VT) que es la técnica de selección de características indicada en la sección 2.3.1 y Linear Support Vector Machine Classification (LSVC) la cual es otra técnica de selección de características que fue descrita en la sección 2.3.2. De la Tabla 4.2 se puede apreciar que este sistema obtiene una mala precisión al no alcanzar siquiera el valor de 0.5 (recordando que el peor valor para todas las métricas utilizadas es 0 y el mejor es 1). Los mejores resultados en este esquema los tuvieron el PS sin selección de características para las 100 y 200 clases mas frecuentes, mientras que para las 500 clases mas frecuentes el PS con selección de características LSVC fue donde se alcanzó el mejor resultado para este sistema.

Tabla 4.2: Precisión promedio SCP

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.11	0.18	0.16	**0.21**	0.10	0.17
200	0.09	0.15	0.14	**0.18**	0.08	0.15
500	0.07	0.13	0.12	0.16	0.13	**0.24**

Curiosamente, y posiblemente enmascarado por la precisión utilizada, los valores de exhaustividad obtenidos por este sistema de clasificación (Tabla 4.3) son idénticos a los de precisión, lo cual es también un resultado pobre.

Los valores F1 obtenidos mediante este esquema de clasificación (Tabla 4.4) muestran la misma tendencia que los resultados anteriores de este clasificador. Los mejores resultados se obtuvieron empleando los vectores de

Tabla 4.3: Exhaustividad promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.11	0.18	0.16	**0.21**	0.10	0.17
200	0.09	0.15	0.14	**0.18**	0.08	0.15
500	0.07	0.13	0.12	0.16	0.13	**0.24**

Tabla 4.4: Valor F1 promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.13	0.22	0.20	**0.25**	0.15	0.21
200	0.11	0.20	0.18	**0.23**	0.13	0.20
500	0.10	0.17	0.15	**0.20**	0.11	0.17

características generados por el preprocesamiento simple.

4.3. Resultados del Sistema Clasificador por Conjunto de clasificadores Naive Bayes

Continuando con el sistema por conjunto de clasificadores empleando clasificadores Naive Bayes. Los acrónimos utilizados son los mismos que los del sistema anterior.

Tabla 4.5: Precisión promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.77	0.81	0.80	**0.82**	0.75	0.77
200	0.71	**0.75**	0.69	0.71	0.69	0.71
500	0.57	0.62	0.62	**0.64**	0.62	**0.64**

En la Tabla 4.5 se ve una mejora considerable con respecto al esquema anterior, esto es, el peor resultado de este sistema supera en 0.33 al mejor resultado del sistema anterior, esta diferencia es incluso mayor que el mejor de los resultado obtenidos por el SBDP, donde la precisión mas baja del SCC es de 0.57 (contra la mas alta del SBDP que es de 0.24). Esta mejora en los resultados ocurre por que se tienen clasificadores dedicados a predecir una etiqueta en particular, en cambio el sistema anterior tiene un solo clasificador realizando cinco predicciones cuando solo fue entrenado con una sola

etiqueta. De igual manera que en el enfoque anterior, el PS (con selección de características como sin estas técnicas), obtiene los mejores resultados.

Tabla 4.6: Exhaustividad promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.52	0.36	0.45	0.32	**0.55**	0.43
200	**0.43**	0.27	0.34	0.24	**0.43**	0.32
500	**0.31**	0.20	0.24	0.16	**0.31**	0.21

Sin embargo, y como lo muestra la Tabla 4.6, el PT con LSVC obtiene la mayor exhaustividad, lo cual indica que este preprocesamiento le permite al clasificador ser menos liberal en sus predicciones.

Tabla 4.7: Valor F1 promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	**0.61**	0.47	0.56	0.42	**0.61**	0.51
200	**0.52**	0.37	0.43	0.31	0.50	0.40
500	**0.39**	0.26	0.32	0.21	0.38	0.27

Los valores de exhaustividad obtenidos ocasionan que el Preprocesamiento Tradicional sea el que tenga mejores valores F1 para todos los conjuntos de clases (Tabla 4.7).

4.4. Resultados del Sistema por Conjunto de Clasificadores SVM

A continuación se presentan los valores obtenidos por el sistema de conjunto de clasificadores, pero empleando clasificadores SVM.

Tabla 4.8: Precisión promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.69	0.74	0.74	**0.75**	0.72	0.74
200	0.62	0.66	0.67	**0.68**	0.66	**0.68**
500	0.53	0.58	0.58	0.59	**0.60**	0.59

La Tabla 4.8 permite apreciar que los clasificadores SVM mostraron un ligero descenso en la precisión con respecto a los resultados obtenidos por los clasificadores Naive Bayes empleados en el mismo esquema (Tabla 4.5), y de manera semejante el Preprocesamiento Simple es el que proporcionó el mayor beneficio a la clasificación salvo para las 500 clases mas frecuentes aunque sin mucha distancia entre el preprocesamiento tradicional con LSVC. La disminución en la precisión (y la poca variabilidad de los resultados entre los diferentes preprocesamientos) indica que los métodos de preprocesamiento no fueron adecuados para las SVM.

Tabla 4.9: Exhaustividad promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.77	0.78	**0.80**	0.79	0.75	0.74
200	0.67	0.68	**0.69**	**0.69**	**0.69**	**0.69**
500	0.59	0.60	**0.61**	**0.61**	0.54	**0.61**

En cambio, la exhaustividad de los clasificadores SVM resulto mayor que la obtenida con los clasificadores Naive Bayes (Tablas 4.7 y 4.10). Estos resultados indican que el SCC utilizando clasificadores SVM genera menos falsos negativos que este mismo sistema utilizando clasificadores Naive Bayes. Sin embargo y de manera similar a lo ocurrido en la Tabla 4.8, es posible apreciarse que los métodos de preprocesamiento generan resultados similares entre si, lo cual corrobora que no fue completamente adecuado el preprocesamiento para los clasificadores SVM.

Tabla 4.10: Valor F1 promedio

clases	PT VT	PT VT	PT	PS	PT LSVC	PS LSVC
100	0.72	0.75	**0.76**	**0.76**	0.73	0.74
200	0.62	0.66	**0.67**	**0.67**	**0.67**	**0.67**
500	0.55	0.58	0.58	**0.59**	0.54	**0.59**

El aumento de los valores de exhaustividad ocasionó que los valores F1 obtenidos mediante los clasificadores SVM (Tabla 4.10) fueran mejores que los expuestos anteriormente para ambos esquemas de clasificación, obteniéndose un valor F1 de 0.76 para las 100 mas frecuentes por parte del PT y el PS, un empate a cuatro entre el PT, PS, PT LSVC y PS LSVC para las 200 clases

mas frecuentes con 0.67 y un empate entre PS y PS LSVC con 0.59 usando las 500 clases mas frecuentes.

4.5. Comparación de los Sistemas de Clasificación

Para comparar los resultados obtenidos por los sistemas de clasificación se emplearon mapas de dispersión donde el eje horizontal es la Exhaustividad (con valores de 0 a 1) y el vertical es la Precisión (con valores de 0 a 1). El mejor resultado posible sería el que se encuentre en las coordenadas (1,1) mientras que el peor se encontraría en las coordenadas (0,0).

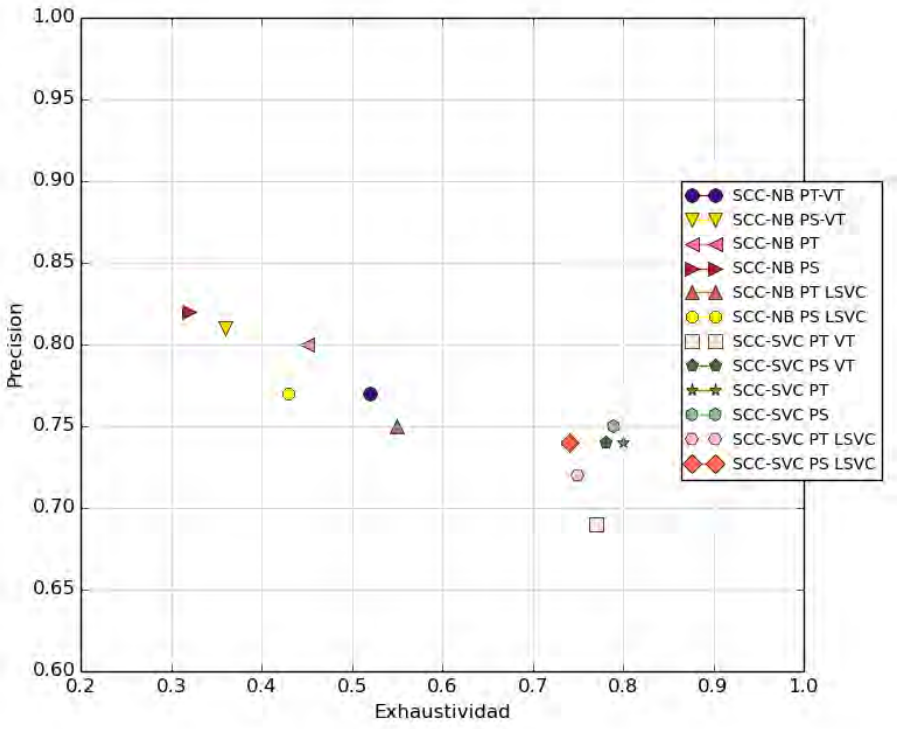

Figura 4.1: Mapa de dispersión para las 100 clases mas frecuentes

En la Figura 4.5 se puede apreciar una ventaja notable de los SCC-NB

sobre los clasificadores SCC-SVC con respecto a la precisión, pero algo pobres respecto a la exhaustividad donde los SCC-SVC se encuentran mas cerca de (1,1) (el mejor valor para ambas métricas es de 1 mientras que el peor es de 0). Curiosamente estos clasificadores se encuentran aglutinados entre si, lo que indica que para las SVM los diferentes esquemas de preprocesamiento no fueron determinantes para la separabilidad de los datos, no así para los SCC-NB.

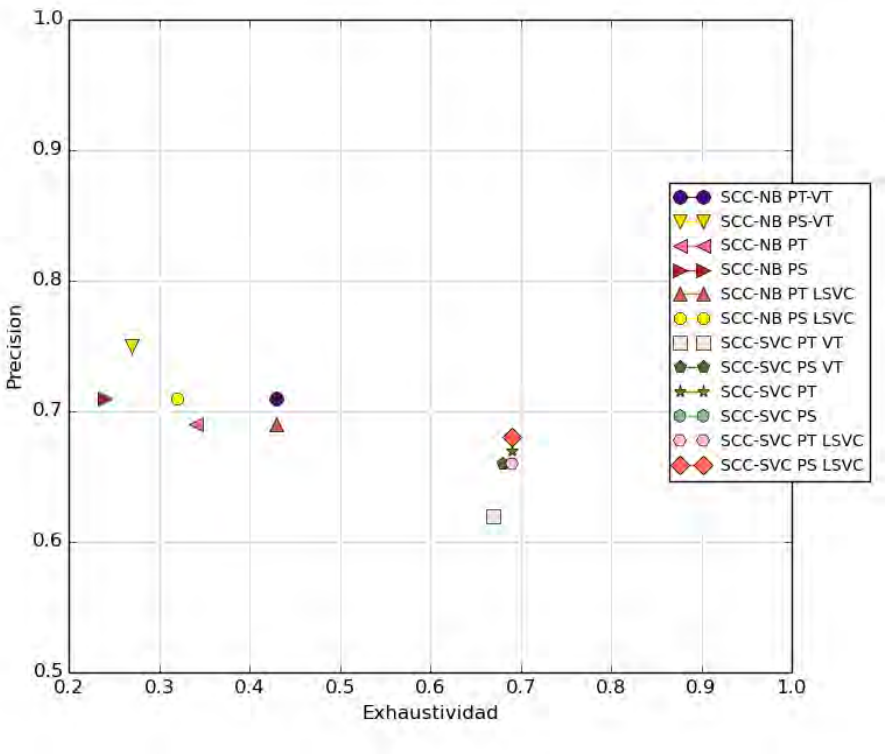

Figura 4.2: Mapa de dispersión para las 200 clases mas frecuentes

La Figura 4.5 muestra un retroceso para todos los clasificadores, principalmente en sus valores de exhaustividad. Esto indica que al aumentar el número de clases a predecir los clasificadores comienzan a ser mas liberales en sus predicciones, esto es, se incrementa el número de falsos negativos. De igual manera que en la Figura 4.5 los clasificadores SCC-SVC son mejores que el resto de los clasificadores.

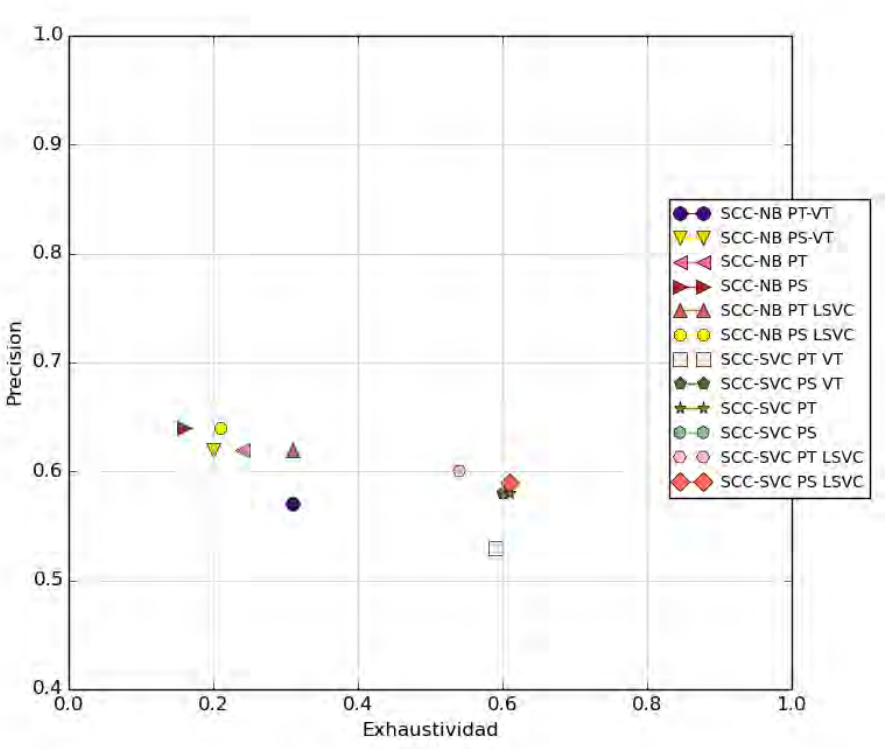

Figura 4.3: Mapa de dispersión para las 500 clases mas frecuentes

En la Figura 4.5 vemos que se incrementa la tendencia observada en la Figura 4.5, lo cual indica que, para estos esquemas de preprocesamiento y clasificación, el incrementar el número de clases de entrenamiento los afecta de manera negativa.

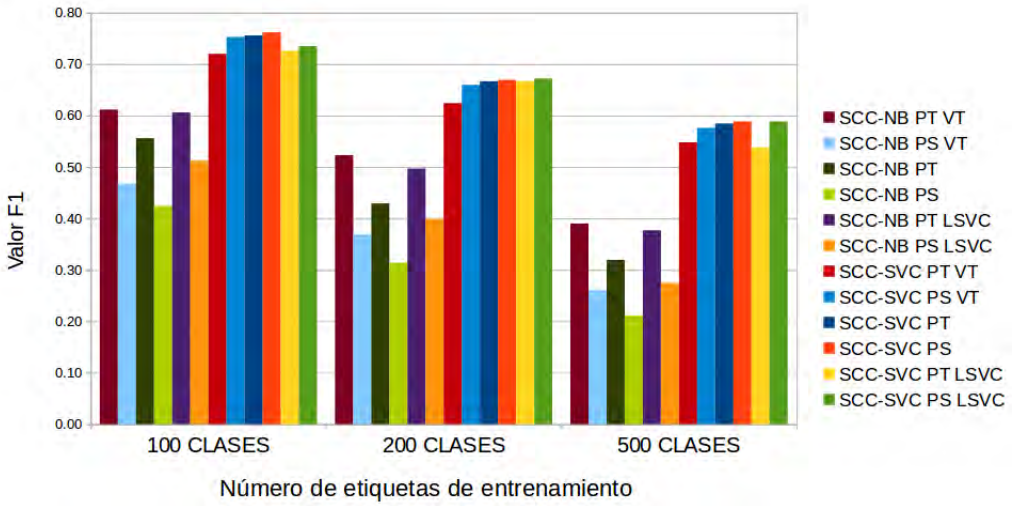

Figura 4.4: Valor F1 promedio por cantidad de etiquetas de entrenamiento

En la Figura 4.5 se observan los valores F1 obtenidos por los diferentes sistemas de clasificación desarrollados. Se puede apreciar que los sistemas SCC poseen los mejores valores F1 destacando para 100 clases los SCC-SVC con PS y PS, para 200 clases todos los SCC-SVC salvo los que usaron VT, y para 500 clases los SCC-SVC con PS y LSVC.

De los resultados obtenidos podemos concluir que el Sistema Clasificador por Conjunto Support Vector Machine Classification proporciona los mejores resultados de exhaustividad y valor F1, en especial si se realiza un Preprocesamiento Simple a los datos. Aunque la precisión alcanzada por los SCC-NB es bastante buena, resultaron muy susceptibles al preprocesamiento, lo que deja la inquietud de generar esquemas de preprocesamiento diferentes para estos sistemas. El rendimiento mostrado por el SBDP resultó decepcionante, aunque en cierta manera predecible debido a su planteamiento sumamente simple, ya que se asumió que el entrenamiento con una sola etiqueta podría inferir 4 etiquetas mas.

4.6. Comparación con otros trabajos

En la sección 1.2 se menciona que no hubo muchas publicaciones que traten sobre este tema. No obstante, es posible comparar los sistemas expuestos en esta tesis contra los desarrollados por Parikh [Parikh, 2013] y

Schuster [Schuster et al., 2013]. La Tabla 4.11 muestra los resultados de estos sistemas y los mejores resultados del sistema basado en clasificador probabilista y el sistema basado en conjunto de clasificadores, ambos para 500 clases.

Tabla 4.11: Comparación contra otros desarrollos

metricas	Parikh	Schuster	SCP	SCC-NB	SCC-SVC
precisión	**0.64**	0.35	0.16	0.57	0.59
exhaustividad	0.25	0.58	0.16	0.31	**0.61**
valor F1	0.36	0.43	0.20	0.39	**0.59**

Aunque el trabajo de Parikh tiene la mejor precisión, la exhaustividad obtenida por su sistema no es lo suficientemente alta para permitirle un mejor valor F1, curiosamente contrario a lo que sucede con el sistema de Schuster. SBDP no es competitivo contra ninguno de los sistemas desarrollados. En contraste, el sistema basado en conjunto de clasificadores SCC-SVC posee el balance adecuado entre precisión y exhaustividad para alcanzar un valor F1 superior con respecto a los demás sistemas (incluido SCC-NB).

Debido a que este trabajo se basó en el concurso Identify keywords and tags from millions of text questions, publicado en kaggle [Goldbloom, 2013] es posible comparar los resultados obtenidos con una amplia gama de desarrollos. En la tabla 4.12 se muestran en orden descendente los 20 mejores valores F1 obtenidos en el concurso[1], así como los nombres de los concursantes. La gran mayoría de los concursantes no publicó trabajo alguno sobre la técnica que emplearon. Los resultados de la tabla fueron calculados usando el conjunto de prueba incluido en el concurso y son los resultados oficiales.

[1]Los resultados completos del concurso se encuentran en el enlace http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/leaderboard/public

Tabla 4.12: 20 mejores valores F1

1	Owen Zhang	0.81382
2	Ruslan Mavlyutov	0.80899
3	E. G. Ortiz-García	0.80626
4	Serge Edunov	0.80517
5	Roberto Díaz Morales	0.80334
6	Alexander Kuznetsov	0.80305
7	Anónimo	0.79836
8	Omar Olmedo	0.79679
9	Anónimo	0.79666
10	Philip Wahrlich	0.79502
11	Gustavo Frigo	0.79474
12	Matt Hagy	0.79131
13	Milton Bose	0.79110
14	Alexander Lapin	0.78564
15	Anton Bogatyy	0.78563
16	Yaser Martínez	0.78561
17	Anónimo	0.78490
18	Shibendu Saha	0.78430
19	Damien Melksham	0.78271
20	Anónimo	0.78202

Desafortunadamente no fue posible completar este sistema antes de que terminara el concurso para poder compararlo de manera directa con el resto de los concursantes, por lo que sólo es posible inferir la efectividad del sistema en base a los resultados mostrados en la sección 4.1.

Obviamente es deseable que los sistemas expuestos en esta tesis hubieran alcanzado mejores puntuaciones, pero debido a las dificultades encontradas al momento de procesar los millones de datos (adecuaciones enfocadas a la administración de la memoria y reducción de tiempos de procesamiento), generaron retrasos que orillaron a probar menos ideas para lograr la clasificación de las preguntas como desarrollar un sistema basado en votación o emplear meta-características.

El valor F1 obtenido por el sistema desarrollado en esta tesis resulta sin embargo competente, considerando que se basa en ideas relativamente sencillas si se compara con los trabajos expuestos en la sección 1.2.

Los resultados expuestos en este capítulo indican que, por el comporta-

miento observado de todas las combinaciones de sistemas de clasificación y selección de características, los esquemas de preprocesamiento desarrollados no fueron completamente adecuados. Aún así, el SCC-SVC fue el que obtuvo mejor valor F1 que los sistemas hechos por Parikh y Schuster.

Capítulo 5

Conclusiones

En esta tesis se ha presentado una serie de sistemas de clasificación multi-clase y multi-etiqueta enfocados a la categorización de texto. Específicamente se ha resuelto el problema de clasificación de preguntas sobre temas computacionales realizadas en el sitio StackOverflow. Estos sistemas fueron diseñados pensando en hacer el menor uso de recursos computacionales y empleando técnicas simples que son fáciles de ser implementadas sin dejar de lado su efectividad para resolver el problema descrito.

5.1. Conclusiones generales

La clasificación multi-clase es un problema complicado y las técnicas desarrolladas suelen atacar el problema de muchas maneras. La forma común es generando clasificadores especializados por cada una de las etiquetas presentes en el conjunto de entrenamiento. Para este caso, este enfoque resulta sumamente lento y costoso a causa de la cantidad de etiquetas presente el cual supera las 40 mil etiquetas; en otros enfoques se entrenaron una fracción de clasificadores [Parikh, 2013] (en su caso 500 clasificadores), los cuales siguen siendo muchos clasificadores por utilizar dada la complejidad adicional que se incorpora al mecanismo de predicción.

En cambio, en este desarrollo con tan solo cinco clasificadores es posible generar resultados competitivos y con ciertas ventajas que no se encuentran disponibles en otros esquemas. Una de estas ventajas es que estos clasificadores pueden ser afinados sobre la etiqueta que intentan predecir. Esto es, si se conoce que existe alta probabilidad de que una etiqueta aparezca en cierta

posición, se puede entrenar al clasificador con cierta parcialidad, cosa que no es posible en otros enfoques como los descritos en la Sección 1.2.

En algunos de los trabajos examinados se dirigieron a un análisis sintáctico de las preguntas para determinar el orden de las etiquetas, cuando existe una tendencia en las mismas etiquetas a ser mas relevantes que otras para la posición a evaluar. En otros desarrollos la predicción está sujeta a parámetros adicionales que agregan *overhead* a la predicción al requerirse evaluaciones adicionales o sub-clasificadores que determinen si la predicción es adecuada o no.

Sin embargo, el principal inconveniente encontrado en el desarrollo de esta tesis fue la caracterización del texto, la cual resulta bastante complicada por factores incluso ajenos a las preguntas en sí, como es la presentación de la pregunta y su formato. El filtrado del código HTML agrega una capa de complejidad extra (por ejemplo: si una pregunta es sobre HTML, ¿Es conveniente eliminar el código HTML de esa pregunta?). La eliminación de términos inusuales, con tanta diversidad de temas discutidos en las preguntas, hace desafiante la separación adecuada de términos. Los mecanismos de selección de características demostraron ser poco efectivos al momento de mejorar los estadísticos de los clasificadores, lo cual es indicativo de que en general las características seleccionadas no aportan la separación suficiente entre las preguntas.

Es conveniente mencionar que aunque este desarrollo es de clasificación de texto, no se utilizó ni análisis sintáctico ni semántico, y sin embargo, fue posible obtener resultados favorables utilizando únicamente las palabras como características. Lo que indica que, si bien esta clase de análisis puede ser de ayuda, no es necesario incorporarlos al tratar con texto.

Como lo demostraron los resultados, no es conveniente utilizar un clasificador simple probabilista para el caso de generar predicciones multi-etiqueta, aunque sería conveniente analizar cuanto le afecta el número de etiquetas de entrenamiento observadas.

El esquema de clasificación con conjunto de clasificadores obtuvo resultados superiores, aunque mejorados principalmente por la exhaustividad (y en menor medida la precisión) alcanzada por los clasificadores de la tercera a la cuarta etiqueta, esto debido a que los clasificadores para estas etiquetas pudieron identificar extremadamente bien a la etiqueta vacía, la cual tiene una presencia alta en estas etiquetas.

Sin embargo lo que resulta claro es que sin una caracterización adecuada no es posible mejorar a los mecanismos de clasificación generados, lo cual se

vio reflejado en el desempeño obtenido.

Los esquemas de solución que mejores resultados obtuvieron fueron los que utilizaron el sistema basado en conjunto de clasificadores utilizando clasificadores SVM y el denominado preprocesamiento simple (con y sin selección de características usando SVC lineal), con valores F1 de 0.76, 0.67 y 0.59 cuando predicen 100, 200 y 500 clases respectivamente. Aunque el SCC con clasificadores Naive Bayes obtuvo la mejor precisión, el bajo resultado que obtuvieron de exhaustividad ocasionó que los valores F1 obtenidos por este sistema fuera inferior al de los SCC-SVC. Se observó que los sistemas SCC-SVC tuvieron resultados muy uniformes en las tres métricas empleadas, lo que indica que los diferentes esquemas de preprocesamiento y de selección de características utilizados no otorgaron separabilidad adicional para estos sistemas. En general, el Sistema Basado en Distribución de Probabilidad obtuvo resultados bastante pobres, debido, muy probablemente, a un entrenamiento inadecuado.

Adicionalmente se compararon estos resultados con los obtenidos por Parikh [Parikh, 2013] y Schuster [Schuster et al., 2013] lográndose una mejora en el valor F1 de 0.23 y 0.16 respecto a sus sistemas utilizando el SCC-SVC. La mejora resulta significativa debido a que el sistema desarrollado en esta tesis posee un mejor balance entre su precisión y exhaustividad que los otros sistemas. Si se compara este sistema con los sistemas concursantes en el concurso Identify keywords and tags from millions of text questions se encuentra ubicado alrededor del 60^{vo} percentil, el cual es un resultado bastante competitivo considerando el número de participantes y las dificultades encontradas.

5.2. Trabajos futuros

Entre las mejoras que se pueden implementar al Sistema Clasificador por Conjunto de Clasificadores es el de adaptar el entrenamiento de los clasificadores de la segunda a la quinta etiqueta, donde el entrenamiento de estos sea en función de las etiquetas mas comunes para esa posición en específico en lugar de las etiquetas mas frecuentes globales. Dado lo observado en los resultados para estas etiquetas, la precisión y exhaustividad general de la segunda etiqueta descienden significativamente. Este comportamiento es influenciado a causa de que las etiquetas mas frecuentes por lo general aparecen en la etiqueta de la primera posición, lo que indica que estos clasificadores

no están observando las etiquetas adecuadas para esa posición.

La caracterización del texto es otro aspecto de mejora importante tanto reduciendo el número de características generadas como la separabilidad entre ellas, ya sea empleando un análisis de lenguaje natural mas sofisticado o bien realizando una mezcla de características y meta-características (como la longitud de la pregunta, frecuencia de palabras clave en la pregunta, la presencia de ciertos caracteres en la pregunta, etc.), lo cual podría beneficiar a los clasificadores que se entrenen bajo este esquema al incorporar elementos totalmente controlables y no solo el contenido de la pregunta.

Sobre el mismo tema de la caracterización del texto se podría aplicar una selección de características basada en árboles de decisión. Estos árboles de decisión realizan la predicción empleando una serie de construcciones lógicas, donde las hojas representan a la clase que pertenece si se cumple toda la serie de consideraciones. Con estos árboles de decisión se podría observar la asociación entre características, lo cual podría ayudar a identificar con mayor claridad las características que mas información proporcionan y así mejorar tanto la clasificación como la incorporación de características a los vectores. Además, al ser un enfoque por conjuntos sería interesante hibridar el mecanismo de clasificación e incorporarle sub-mecanismos de clasificación. Al procesarse una pregunta se pudiera examinar si el texto posee desde un inicio alguna de las etiquetas ya sea en el título o bien el cuerpo de la pregunta para seleccionarla directamente como etiqueta y reducir el número de predicciones por parte de los clasificadores.

Referencias

E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. ISBN 9780262012430. URL `http://books.google.com.mx/books?id=4j9GAQAAIAAJ`.

Francesc Alted, Ivan Vilata, et al. PyTables: Hierarchical datasets in Python, 2002. URL `http://www.pytables.org/`.

Jeff Attwood. StackOverflow. `http://www.stackoverflow.com`, 2013. [Online; accessed 13-November-2013].

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Iain S Duff, Roger G Grimes, and John G Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)*, 15(1):1–14, 1989.

Anthony Goldbloom. Kaggle. `http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction`, 2013. [Online; accessed 13-November-2013].

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

James Hong and Michael Fang. Keyword extraction and semantic tag prediction. 2013.

Han Jiawei and Micheline Kamber. Data mining: concepts and techniques. *San Francisco, CA, itd: Morgan Kaufmann*, 5, 2001.

Min ling Zhang and Zhi hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *PATTERN RECOGNITION*, 40:2007, 2007.

Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104, 2012.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

Chintan Parikh. Identifying tags from millions of text question. 2013.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

Leonard Richardson. Beautiful soup, 2004. URL `http://www.crummy.com/software/BeautifulSoup/`.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

Lorenzo Rosasco, E Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

E.R. Scheinerman. *Matemáticas discretas*. Thomson, 2001. ISBN 9789706860712. URL `http://books.google.com.mx/books?id=y9sgAAAACAAJ`.

Sebastian Schuster, Wanying Zhu, and Yiying Cheng. Predicting tags for stackoverflow questions. 2013.

Clayton Stanley and Michael D Byrne. Predicting tags for stackoverflow posts. In *Proceedings of ICCM*, 2013.

Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Machine Learning: ECML 2007*, pages 406–417. Springer, 2007.

S. van der Walt, S.C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2011.37.

Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1338–1351, 2006.

Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani. 1-norm support vector machines. *Advances in neural information processing systems*, 16 (1):49–56, 2004.