UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

COMBINADOR DE CLASIFICADORES PARA IDENTIFICAR MEMES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

NICOLÁS FRANCISCO MACHUCA GUERRA

PROFESOR GUÍA:
BENJAMÍN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:
FELIPE BRAVO MÁRQUEZ
JOSÉ BENGURIA DONOSO

SANTIAGO DE CHILE
2021

Resumen

La clasificación de imágenes es un estudio que se ha vuelto muy común en la computación, sobre todo en los últimos años, impulsado por la aparición de grandes repositorios de imágenes de los cuales hacer uso. Dentro de la clasificación de imágenes existen una cantidad casi ilimitada de temas en los que enfocar el estudio, y los memes, siendo una de las formas de imágenes más masificadas en la actualidad, serán el principal enfoque de estudio en este trabajo, apoyado por un dataset de imágenes provisto por Instituto Milenio Fundamentos de los Datos (IMFD).

En el estado del arte se presentan algunos resultados para el problema de clasificar una imagen en las clases No Meme, Meme y Sticker, estos resultados muestran que el mejor resultado obtiene una precisión cercana al 75 %. Además de este resultado, también indica que si se tuviera un *oráculo* que pudiera elegir el mejor modelo de clasificación para cada imagen, en el 99 % de los casos se obtendría una predicción correcta, este resultado es la principal motivación para este trabajo, siendo el objetivo principal, buscar formas de combinar los modelos ya existente, con el fin de superar el mejor resultado individual, el cual alcanza el 75 % de precisión.

Los resultados presentados en el estado del arte, en el trabajo de memoria "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020, utilizan el método *undersampling* para preprocesar el conjunto de datos de entrenamiento y equilibrar la cantidad de ejemplos de cada una de las clases. Además de esta forma de estudiar este problema de clasificación, en este trabajo también se plantea un nuevo experimento, tanto para los modelos individuales como también para las combinaciones de ellos, basado en la técnica de entrenamiento *cost-sensitive*, buscando aprovechar la totalidad del dataset disponible, el cual presenta un enorme desbalance en los ejemplos para cada una de las clases.

En el caso de experimentación usando *undersampling*, los resultados obtenidos con los métodos de ensemble, en los mejores casos, muestran una leve mejora en comparación al mejor resultado de los modelos individuales del estado del arte, pasando de un 75 % de precisión a un 77 % y obteniéndose una disminución en el error porcentual en el cálculo de las métricas. Para la experimentación usando *cost-sensitive*, los modelos individuales entregan como mejores resultados los obtenidos por la combinación conformada por el descriptor DeCAF7 y el clasificador *Random Forest*, por otro lado, en los ensembles, los mejores resultados se obtuvieron al usar el método de Votación Mayoritaria con Peso, que si bien reduce la precisión en comparación al mejor modelo individual, aumenta en gran medida el valor del *recall*.

Para todos aquellos que me enseñaron
a transitar el camino de la vida

Saludos

Agradecimientos

Parto agradeciendo a quienes me proporcionaron las bases de este trabajo de memoria: primero, a mi profesor guía Benjamín Bustos, quien estableció la idea base de este trabajo, además de proporcionarme toda la ayuda necesaria, para realizar de la mejor forma esta memoria; segundo, a Alberto Sara Zaror, quien me proveyó del material necesario para iniciar mi trabajo, tanto códigos como bases de datos, y por su disposición a resolver mis dudas con respecto a este material; por último, agradecer al Instituto Milenio Fundamentos de los Datos (IMFD) por ser la fuente de la base de datos usada en este trabajo.

También, quiero agradecer a mis amigos más cercanos, partiendo por mis compañeros de universidad, Toño y Bryan, quienes fueron un fuerte pilar en mi paso por la universidad, ayudándome, sobre todo en las situaciones académicas donde requería apoyo. Por otro lado, agradezco al *Krosty*, al *Androito*, al *Game* y al resto de cabros de discord por estar presentes para hablar, jugar o hacer cualquier cosa para hacer más ameno el encierro durante esta cuarentena.

Por último, quiero agradecer a las personas más importantes a lo largo de mi existencia, mis padres y hermanas, quienes siempre me han apoyado y me han impulsado en cada paso que he dado para llegar a este hermoso punto en mi vida.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Junto con el gran crecimiento tecnológico de los últimos tiempos, han surgido varias formas nuevas de comunicación y de expresión, pero muy pocas han sido tan masificadas e influyentes como los memes. Nacidos en Internet con la finalidad de entretener, los memes usualmente compuestos de una imagen y texto superpuesto, hoy en día se encuentran presentes en gran parte de la vida de las personas, convirtiéndose en una popular forma de expresión. Es por ello que realizar estudios que involucren este tipo de imágenes puede llevar a resultados interesantes y justamente eso es lo que se intentará lograr en este trabajo de memoria.

1.1. Motivación

Debido a la gran masificación de los memes, y su extendido uso, sobre todo en redes sociales, existe una cantidad casi ilimitada de memes distintos, esto permite realizar estudios sobre el tema con una gran cantidad de ejemplos y con ello conseguir una mayor validez en los resultados obtenidos. Gracias a esto, en el trabajo de memoria "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020 [1], se realiza un estudio para evaluar el desempeño de distintos descriptores visuales combinados con clasificadores del estado del arte puestos a prueba en el problema de clasificar una imagen en tres clases: No-Meme, Meme y Sticker (Algunos ejemplos de esas clases se pueden ver en la Figura 1.1).

a No Meme b Meme c Sticker

Figura 1.1: Ejemplos de cada una de las clases No Meme, Meme y Sticker

Un resultado interesante al que se llega en el trabajo de memoria ya mencionado, es que aunque la mejor combinación descriptor-clasificador logra una precisión del 75 %, al probar todos los modelos, en el 99 % de las imágenes de prueba al menos una las combinaciones predice correctamente la clase a la cual pertenece la imagen. Este resultado puede ser un indicativo de que al usar los modelos en conjunto, por ejemplo a través de ensembles, es posible obtener mejores precisiones que las obtenidas por cada una de las combinaciones descriptor-clasificador probadas de forma individual.

Los modelos usados en el trabajo mencionado, para este caso de clasificación, son entrenados con un dataset preprocesado con la técnica de *undersampling*, lo cual facilita el análisis de desempeño, pero desaprovecha gran parte del conjunto de ejemplos del cual se dispone, es por ello que, en este trabajo, se realizaran dos experimentos, el primero ocupará la técnica ya mencionada, *undersampling* y el segundo se enfocará en usar todo el dataset a través de la técnica de entrenamiento *cost-sensitive*, la cual permite mejorar el entrenamiento de clasificadores en conjuntos de datos donde existe un desbalance en los ejemplos disponibles para cada una de las clases.

1.2. Objetivos

Siendo la principal motivación de este trabajo, usar los resultados ya obtenidos de los modelos conformados por la combinación de un descriptor visual y un clasificador, a través de ensembles para lograr un trabajo conjunto de estos modelos, el objetivo principal que guía el desarrollo de este proyecto es evaluar el desempeño de distintos ensembles que utilicen los modelos disponibles, buscando obtener una mejora en los resultados alcanzados en el trabajo de memoria "Comparación de descriptores para clasificación de memes".

Para lograr el objetivo principal, se plantean los siguientes objetivos específicos a cumplir:

1. Investigar y plantear distintas técnicas para combinar los modelos descriptor-clasificador.
2. Definir medidas cuantitativas para establecer la eficacia de los métodos planteados.
3. Implementar las técnicas planteadas.
4. Entrenar, de ser necesario, las técnicas implementadas.
5. Obtener y analizar los resultados de probar las técnicas implementadas

Capítulo 2

Antecedentes

En este capítulo se revisaran algunos conceptos y/o herramientas del estado del arte que permitirán un mejor entendimiento de lo que se realizará y los resultados que se obtendrán en este trabajo.

2.1. Clasificación de imágenes

Siendo el punto de partida primordial para este trabajo, la clasificación de imágenes ha sido un tema de frecuente estudio en la computación, impulsada, en los últimos años por el explosivo crecimiento del aprendizaje de máquina (*Machine Learning*) y la constante generación de nuevo material, principalmente a través de internet.

La gran variedad de métodos que existen en el estado del arte que buscan solucionar el problema de la clasificación también se ven reflejados en la clasificación de imágenes, sumado a esto una imagen entrega mucha información y de distinto tipo, ya sea las variadas formas, texturas y colores presente, y otros datos mas abstractos y difíciles de analizar como las expresiones faciales, objetos e incluso el contenido textual que pueda incluir. Todo esto crea una variedad casi ilimitada de puntos de enfoque que usar al momento de clasificar imágenes.

Este trabajo se enfocara en clasificar imágenes en memes, sticker o no memes, para ello se tomara en cuenta las características visuales de las imágenes usando modelos descriptor-clasificador [1] presentes en el estado del arte.

2.2. Propiedades de imágenes

Siendo las imágenes el elemento base en este trabajo de memoria, es necesario definir ciertas características relevantes que poseen.

Las imágenes más comunes y de mayor utilidad para este trabajo son las imágenes RGB. Estas son representadas como una matriz tridimensional, es decir, una matriz donde cada celda contiene tres valores numéricos. Cada celda representa un píxel de la imagen y cada valor de la celda la intensidad de los colores primarios rojo (*red*), verde (*green*) o azul (*blue*) presentes en el píxel. De esta forma, como matriz, una imagen RGB tiene tamaño $N \times M \times 3$ con N el alto y M el ancho (en píxeles) de la imagen.

Dado que las imágenes son representadas como matrices es posible aplicarle algunas transformaciones que serán de utilidad para este trabajo. Las dos principales son: Transformación a escala de grises y Partición.

Transformación a Escala de Grises

Esta transformación convierte la matriz representante de la imagen de forma que se pierde la tridimensionalidad quedando cada celda como un valor numérico único que representa la tonalidad de gris del píxel. Para lograr la transformación, en cada celda se pondera los valores de los tres canales de RGB. El resultado se conoce como una imagen en escala de grises, un ejemplo de ello se puede observar en la Figura 2.1.

a Imagen Original b Imagen en escala de grises

Figura 2.1: Transformación a escala de grises

Partición

Este proceso se busca dividir la imagen en distintos bloques de igual tamaño, para ello se definen n y m como la cantidad de divisiones verticales y horizontales de la imagen respectivamente.

a Imagen Original

b Grilla de 8×8 sobre la imagen origina

Figura 2.2: Partición de una Imagen

Sea I una imagen en espacio de color RGB de tamaño $N \times M \times 3$. Se divide cada canal de I con una grilla de $n \times m$, cada celda de esta grilla es referida como I_{uvc}, en donde los subíndices u y v representan la fila y columna de la grilla y c el canal. El resultado de este proceso se puede ver en la Figura 2.2.

2.3. Descriptores Visuales

Los descriptores visuales son algoritmos que permiten parametrizar las características visuales de una imagen, convirtiendo estas características en información más fácil de interpretar para el computador y, en este trabajo, para la posterior lectura por parte de los clasificadores.

Un descriptor visual usualmente se enfoca en una cierta característica de la imagen, esto impulsa la existencia de una gran cantidad de descriptores diferentes. De forma general se tienden a clasificar en dos tipos:

- **Descriptores de información general:** Como bien indica su nombre este tipo de descriptores se enfocan en la parametrización de elementos generales presentes en todas las imágenes como los colores, formas, texturas, bordes, etc.
- **Descriptores de información de dominio específico:** Son descriptores de más alto nivel que usualmente se apoyan en los descriptores anteriores para obtener información de los objetos y eventos que conforman una imagen. Son utilizados comúnmente para identificar objetos presentes en una imagen e incluso tienen una gran importancia en el reconocimiento facial.

Este trabajo se centrara en el uso de los descriptores de bajo nivel, los **Descriptores de información general**, se utilizaran cuatro descriptores de esta rama:

- **Distribución de Color**
- **Histograma de Grises**
- **Histograma de Bordes**
- **Histograma de Gradientes Orientados**

2.3.1. Distribución de Color

Como bien indica su nombre, Distribución de Color [2] es un descriptor que se encarga de crear un vector de características que representa la distribución espacial de los colores en la imagen. Para lograr esto, primero es necesario simplificar la imagen, dado que estas pueden tener una gran cantidad de colores y píxeles lo que convertiría esta tarea en algo sumamente complicado. Para hacer más fácil este proceso de descripción se aplican dos transformaciones a la imagen, primero se particiona y luego se obtiene el color representativo de cada una de las partes. Para terminar la descripción se transforma el espacio de colores original (RGB) a YCbCr y se aplica la transformada discreta del coseno.

Color Representativo

Luego de obtener la partición de la imagen original en una grilla de 8×8 se calcula, para cada uno de los bloques, el color representativo. Se puede usar cualquier método que sirva para obtener un color representativo, pero de forma estándar y para efectos de este trabajo se usa el promedio de los colores de cada píxel perteneciente al bloque (ver Figura 2.3).

a Imagen Original

b Color representativo con partición 8×8

Figura 2.3: Color representativo

Transformada Discreta del Coseno

Sea I' la transformación de una imagen en espacio de color RGB a espacio YCbCr. Se define C[1] una matriz con componentes igual a:

$$C_{jk}^{(N)} = \sqrt{\frac{\alpha_j}{N}} \cos\left(\frac{\pi(2k+1)j}{2N}\right)$$

donde N es el número de filas de la imagen I', j y k son las componentes de la matriz C y α_j se define como:

$$\alpha_j = \begin{cases} 1 & j = 0 \\ 2 & j > 0 \end{cases}$$

Con C definida, se establece la transformada discreta del coseno como:

$$TDC(I) = C^{(N)} \cdot I \cdot (C^{(N)})^T$$

Vector de Características

Para la obtención del vector de características, se calcula el promedio de la transformada discreta del coseno por cada una de las celdas de la grilla, para ello se genera un matriz T cuyas componentes se definen como:

$$T_{ijc} = prom(TDC(I'_{ijc}))$$

donde I'_{ijc} es la componente i, j de la grilla que particiona a I' y c es el canal al cual se le calcula la transformada discreta del coseno.

Finalmente el vector de características de este descriptor se genera concatenando los elementos de la matriz T, correspondiente a cada componente de la grilla y los tres canales de colores presentes. El vector de características D se define como:

$$D = (T_{000} \cup T_{010} \cup ... \cup T_{080} \cup T_{180} \cup ... \cup T_{880} \cup T_{001} \cup ... \cup T_{883})$$

2.3.2. Histograma de Grises

Como descriptor visual el Histograma de Grises busca representar, en un vector de características, la frecuencia de las distintas tonalidades de grises presentes en una imagen con un espacio de color en escala de grises. De esta forma se establece como primer paso, convertir la imagen a escala de grises, luego, de igual forma como se realiza en el descriptor de distribución de color, se realiza una partición de la imagen (ver Figura 2.2) y a cada una de las particiones se le obtiene su propio histograma de grises. Por último, el resultado final de este descriptor viene dado por la concatenación de los histogramas de cada una de las partes de la imagen.

[1] https://docs.rs/opencv/0.19.2/opencv/core/fn.dct.html

Histograma de Grises por Celda

Sea I una imagen en escala de grises de tamaño $N \times M$, se divide I con una grilla de $U \times V$. Sea I_{uv} la celda ubicada en la fila u y columna v de la grilla que divide a I.

El rango de valores que puede tener un píxel en una imagen en escala de grises varía entre 0 y 255, es decir, para cada celda de la imagen es necesario calcular 256 frecuencias para obtener el histograma de grises, esto provoca que según el tamaño de la grilla el resultado final pueda tener un tamaño considerable. Para solucionar este problema se define un número B denominado bins el cual define el número de sub partes en las que dividir el rango de valores inicial ($[0, 255]$). Con $B = 8$ la división quedaría:

$$[0, 255] = [0, 31] \cup [32, 63] \cup .... \cup [224, 255]$$
$$[0, 255] = bin_0 \cup bin_1 \cup .... \cup bin_7$$

Con esta nueva división, el histograma H_{uv} para la celda I_{uv} se calcula como un arreglo de frecuencias normalizadas correspondientes a cada uno de los sub rangos de la escala de grises.

Sea F_{buv} la frecuencia normalizada del sub rango b de la escala de valores en la celda I_{uv}, entonces el histograma de grises para la celda I_{uv} se calcula como:

$$H_{uv} = (F_{0uv}, F_{1uv}, ..., F_{B-1uv})$$

Histograma de Grises de la Imagen

Habiendo obtenido el histograma de grises para cada una de las celdas en la grilla, el descriptor de grises de la imagen completa se calcula como la concatenación de cada Histograma H_{uv}, es decir:

$$D = (H_{00} \cup H_{10} \cup ... \cup H_{U0} \cup H_{01} \cup ... \cup H_{0V} \cup ... \cup H_{UV})$$

2.3.3. Histograma de Bordes

El Histograma de Bordes [3] busca representar la frecuencia de ciertos tipos de bordes en la imagen. Estos bordes se identifican a través de la convulsión entre la imagen transformada a escala de grises y un grupo de kernels donde cada uno permite detectar un tipo de borde específico.

Kernels

Para la identificación de cada uno de los tipos bordes con los que trabajara este histograma, se definen los siguientes kernels:

$$K_1 = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \quad K_2 = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \quad K_3 = \begin{pmatrix} \sqrt{2} & 0 \\ 0 & -\sqrt{2} \end{pmatrix} \quad K_4 = \begin{pmatrix} 0 & \sqrt{2} \\ -\sqrt{2} & 0 \end{pmatrix} \quad K_5 = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$

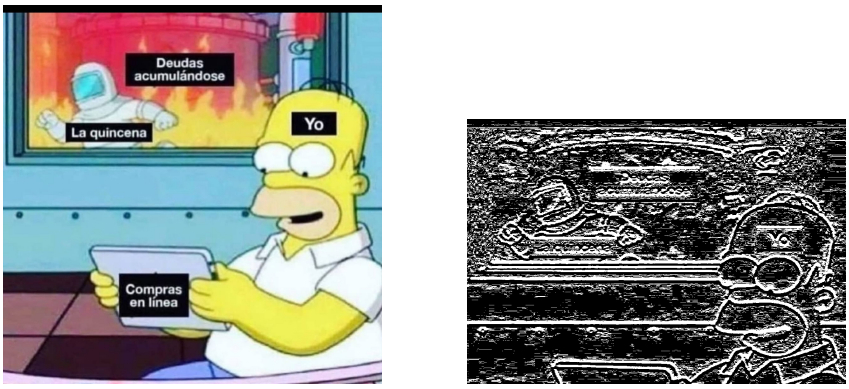
De esta forma, cada uno de estos kernels permite resaltar, en una imagen en escala de grises, un tipo de borde específico. En este caso, los kernels están diseñados para obtener los bordes según su dirección, por ejemplo, K_1 permite obtener los bordes horizontales mientras que K_2 los bordes verticales.

Convolución

Sea I una imagen en escala de grises de tamaño $N \times M$. Se define el conjunto I'_K como la convolución entre I y cada uno de los kernels definidos previamente. Es decir:

$$I'_K = \{I_n \in I'_K \mid I_n = I * K_n\}$$

donde $*$ representa la operación convolución y I_j una matriz que representa los bordes extraídos en cada punto de la imagen I usando el kernel K_j, un ejemplo del resultado de este proceso se puede observar en la Figura 2.4.



a Imagen Original

b Obtención de Bordes por convolución con el kernel K_1

Figura 2.4: Bordes de una Imagen

Borde Representante

Se define la matriz B_D como:

$$B_D(x, y) = argmax([\ |i_1(x, y)|, ..., |i_5(x, y)|\ ])$$

donde $argmax$ es una función que obtiene el índice donde se encuentra el máximo valor dentro de un arreglo y, por ende, $B_D(x, y)$ representa el índice del kernel que maximiza el valor absoluto, tras la convolución con la imagen I, en el punto $(x,\ y)$. Es decir, B_D identifica el tipo de borde que predomina en cada punto de la imagen.

Histograma de la Imagen

Para finalizar el proceso de descripción de la imagen, se divide B_D con una grilla de tamaño $U \times V$. Luego, a cada celda B_{Duv} de la grilla se le calcula un histograma H_{uv}, cuya dimensionalidad es igual al número de kernels usados. El resultado final de este descriptor se define como:

$$D = (H_{00} \cup H_{10} \cup ... \cup H_{U0} H_{01} \cup ... \cup H_{0V} \cup ... \cup H_{UV})$$

2.3.4. Histograma de Gradientes Orientados

El Histograma de Gradientes Orientados [4] permite transformar una imagen en un vector de características, el cual representa la frecuencia, en distintos sectores de la imagen, de los gradientes orientados. Los gradientes orientados miden como cambian los colores y sus intensidades en una imagen, siendo definidos, en cada punto de la imagen, como un vector bidimensional que representa la intensidad y ángulo en la que cambia el color en una porción de la imagen.

Gradientes horizontales y verticales

Para obtener el gradiente en un punto, es necesario primero obtener el gradiente vertical y el horizontal de ese punto, para ello se hace uso de los siguientes dos kernel:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad K_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 3 \end{pmatrix}$$

Se define I como una imagen en escala de grises de tamaño $N \times M$. Usando K_x y K_y se calcula:

$$I_x = K_x * I \qquad I_y = K_y * I$$

donde I_x representa los gradientes horizontales y I_y los gradientes verticales en cada uno de los puntos de la imagen I.

Magnitud y Ángulo de los Gradientes

Posterior a la obtención de los gradientes, tanto verticales como horizontales, se calcula la magnitud de estos en cada punto de la imagen, para ello se define I_M como:

$$I_M = \sqrt{I_x^2 + I_y^2}$$

Además, se obtiene la matriz que representa la dirección de los gradientes en cada punto. Se define I_Θ como:

$$I_\Theta = \arctan\left(\frac{I_y}{I_x}\right)$$

Histograma

Sea I una imagen en escala de grises de tamaño $N \times M$, se divide I con una grilla de $U \times V$. Sea I_{uv} la celda ubicada en la fila u y columna v de la grilla que divide a I.

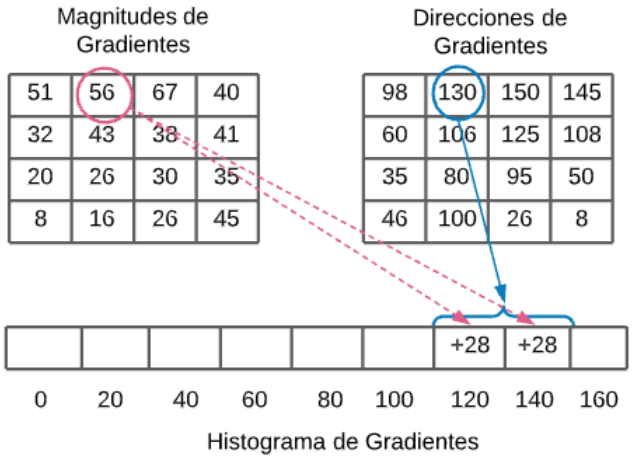


Figura 2.5: Ejemplo de obtención del Histograma de Gradientes Orientados

El cálculo del histograma busca representar la frecuencia de los ángulos de los gradientes por celda tomando como medida de frecuencia su magnitud, para ello las orientaciones se pueden presentar en el rango de 0° a 180°, lo que consideraría los ángulos con signo opuesto como el mismo, o en el rango de 0° a 360°. Para el primer caso el histograma, generalmente se calcula como un vector de largo 9, cada espacio representa un ángulo, separados por una distancia equitativa $(0, 20, 40, ..., 160)$, donde se agrega la magnitud correspondiente, en caso de que un gradiente tenga una orientación intermedia, su magnitud se divide dando una mayor parte a la celda correspondiente al valor más cercano, un ejemplo de este proceso se puede observar en la Figura 2.5. En el caso del rango 0° a 360°, el único cambio que surge en que el largo del histograma aumenta a 18 conservando los intervalos y aumentándolos hasta 340°.

Histograma de la Imagen

Finalmente, y como ya se estudió en los descriptores previos, el resultado que entrega el descriptor visual es la concatenación de los histogramas de cada una de las celdas en las que se dividió la imagen. Es decir, se entrega el vector de características D como:

$$D = (H_{00} \cup H_{10} \cup ... \cup H_{U0} H_{01} \cup ... \cup H_{0V} \cup ... \cup H_{UV})$$

2.3.5. Descriptores basados en Redes Convolucionales

Las Redes Convolucionales son modelos de clasificación basados en los perceptrones multicapas [5]. Mientras que un perceptrón multicapa utiliza un vector unidimensional de pesos entrenables para procesar el dataset, las redes convulucionales utilizan una matriz bidimensional, denominada kernel, y la operación convolución para transformar los datos en cada una

de las capas. Estos kernels se caracterizan por ser de un tamaño reducido e independiente del tamaño del input, de esta forma se pueden procesar imágenes de distinto tamaño sin tener que utilizar una capa de entrada con pesos para cada uno de sus píxeles. Son estos mismos kernels los que permiten extraer características de la imagen (de forma similar a como se observó en el Capítulo 2.3.3).

Usualmente, las últimas capas de las redes convolucionales están formadas por perceptrones, es decir, vectores unidimensionales de pesos. Estas capas son las que, finalmente permiten realizar la clasificación, utilizando las características extraídas por las capas convolucionales formadas por kernels.

De esta forma, las redes convolucionales pueden ser usadas como descriptores, para ello se extraen las capas de clasificación y se utiliza como vector de características la salida de la última capa formada por kernels. En comparación con los descriptores clásicos mencionados anteriormente donde las características que se buscaban parametrizar se dejaban establecidas desde el comienzo, al utilizar una red convolucional es difícil determinar cuál es su enfoque en la extracción de características, dependiendo tanto de la arquitectura de la red como también del dataset de entrenamiento.

DeCAF7

Deep Convolutional Activation Features (DeCAF) [6] es una red neuronal convolucional diseñada para extraer características de imágenes. Una de sus principales cualidades es que fue entrenada como un clasificador sobre el ImageNet Large Scale Visual Recognition Challenge 2012, el cual posee una variedad de más de 10 millones de imágenes que se clasifican en 10000 clases distintas.

Para su uso como un descriptor visual, la arquitectura de DeCAF consta de 7 capas convolucionales. Se denota como $DeCAF_n$, al uso de DeCAF para extraer características de una imagen utilizando las n primeras capas convolucionales de su estructura, de esta forma se recomienda usar $DeCAF_5$, $DeCAF_6$ o $DeCAF_7$ como descriptores visuales, debido a que usar una capa previa podría no entregar características de interés de la imagen.

La eficacia de DeCAF como descriptor visual fue constatada al entrenar un máquina de vectores de soporte utilizando $DeCAF_6$ en el dataset Caltech-101. Este experimento arrojó una precisión de 86.9 %.

ResNet152

Residual Neural Network (ResNet) [7] es una red neuronal, la cual gracias a su optimizado proceso de entrenamiento, permite mantener una alta profundidad sin perder precisión. En una red normal tener una alta profundidad se vuelve insostenible debido a que el error que se obtiene en la capa de salida se debe propagar al resto de capas, las cuales procesan este valor, resultando en que, para las primeras capas, las optimizaciones basadas en la reducción del error no sean las mejores.

Para lograr una alta optimización con una red neuronal profunda, ResNet hace uso de una función residual. Sea $\mathcal{H}(x)$ la salida de un subconjunto de capas del modelo ante una

entrada de datos x, se puede definir el error correspondiente a $\mathcal{H}(x)$ como $\mathcal{F}(x) = \mathcal{H}(x) - x$. Con esto se puede definir la salida y como:

$$y = \mathcal{F}(x, W_i) + x$$

donde $\mathcal{F}(x, W_i)$ es la función residual que representa el error al aprender en la capa i, x e y son las entradas y salidas de la capa. Al calcular la operación $\mathcal{F} + x$, en cualquier capa, se puede conectar directamente la entrada a la salida de la capa, de esta forma no se agregan nuevos parámetros y tampoco se aumenta la complejidad.

ResNet también se puede utilizar como un descriptor visual utilizando solo las capas convolucionales para extraer un vector de características de la imagen y luego entrenar otro tipo de clasificador.

2.4. Clasificadores

Los clasificadores permiten recibir la información parametrizada de una imagen, usualmente como un vector de características y con ello predecir la clase a la que pertenece la imagen.

En el estado del arte existen una gran cantidad de métodos clasificadores, en general estos se separan en dos grandes grupos, de aprendizaje supervisado y de aprendizaje no supervisado:

- **Aprendizaje Supervisado**: Esta clase de aprendizaje requiere que los datos de entrenamiento estén acompañados con su respectivo resultado deseado. De esta un clasificador supervisado es guiado a través del set de datos de entrenamiento para que entregue una respuesta acorde con lo que el usuario requiere.
- **Aprendizaje No Supervisado**: Esta clase de aprendizaje no requiere que los datos de entrenamiento se entreguen con sus valores deseados, mas bien son los clasificadores de este tipo los que descubren, a través de los datos de entrenamiento, características, correlaciones, categorías, etc. entre los distintos elementos del set de datos.

En este trabajo de memoria el enfoque se centra en los modelos clasificadores de aprendizaje supervisado, dado que los datos de ejemplos ya se encuentran etiquetados y los resultados que se espera que cada modelo entregue ya se encuentran delimitados por las clases meme, no meme y sticker.

2.4.1. Máquina de Vectores de Soporte

El modelo de Máquina de Vectores de Soporte (SVM) [8] permite resolver el problema de la clasificación modelando cada uno de los ejemplos de entrada como puntos en un espacio de alta dimensionalidad, el modelo se entrena para buscar el mejor hiperplano que logre separar las distintas de forma que se pueda determinar la clase a la que pertenece un ejemplo a través de su posición en el espacio con respecto al hiperplano separador.

Sea $x \in X$ un vector de características de tamaño p, donde X es el conjunto de datos de entrada para el modelo. Se define el hiperplano H, perteneciente a un espacio de dimensionalidad p, como:

$$H = b_0 + b_1 x_1 + ... + b_p x_p$$

donde cada uno de los b_i son parámetros que definen el hiperplano. De esta forma x' pertenece al hiperplano H si se cumple la ecuación:

$$b_0 + b_1 x'_1 + ... + b_p x'_p = 0$$

Se define el conjunto de entrenamiento como $(x_i, y_i)_{i=1...n}$, con $x_i \in X$ elemento del conjunto de datos de entrada y $y_i \in \{-1, 1\}$ clase asignada al elemento x_i. Entonces el hiperplano separador debe cumplir:

$$b_0 + b_1 x_{i1} + ... + b_p x_{ip} < 0, \;\; si \;\; y_i = -1$$

$$b_0 + b_1 x_{i1} + ... + b_p x_{ip} > 0, \;\; si \;\; y_i = 1$$

ambas condiciones pueden simplificarse en:

$$y_i(b_0 + b_1 x_{i1} + ... + b_p x_{ip}) > 0, \;\; para \; i = 1...n$$

De esta forma, un elemento x es clasificado según el signo de la función $f(x) = b_0 + b_1 x_1 + ... + b_p x_p$, siendo el hiperplano lo que se debe optimizar en la fase de entrenamiento, buscando el que se conoce como *hiperplano óptimo de separación*, aquel que se encuentran más alejado de todos elementos del conjunto de entrenamiento y que mantiene la separación de las clases. Este método es efectivo, pero solo en casos en el que las clases sean perfectamente separables linealmente, esto pocas veces ocurre en un ejemplo real, es por eso que surge el Clasificador de Vectores de Soporte (SVC), el cual permite que el hiperplano tenga algunas fallas al separar el espacio, a cambio de un aumento en la robustez y la capacidad predictiva. En SVC se define un parámetro C el cual indica cuan riguroso es el modelo ante un fallo, es decir, si $C = \infty$ se obtiene el caso base donde el modelo no permite fallos y se busca el hiperplano óptimo de separación, entre más cercano a cero el valor de C menos se penalizan los errores, lo que implica que más ejemplos del set de entrenamiento pueden estar posicionados de manera errónea con respecto al hiperplano.

2.4.2. Random Forest

Random Forest [9] es un método de clasificación que basa su funcionamiento de en el trabajo en conjunto de varios árboles de decisión, donde cada árbol es entrenado con datos independientes y con la misma distribución en sus clases. Al momento de realizar una predicción cada uno de los árboles de decisión entrega su resultado independiente del resto, la respuesta final en este método de clasificación viene dada por el promedio o votación mayoritaria de lo entregado por cada uno de los árboles de decisión.

Sea $(x_i, y_i) \in T$ pares del conjunto de entrenamiento. Sea B el número de árboles de decisión presentes en el random forest, T_b un conjunto de tamaño n formado por la obtención con reposición de elementos de T y que servirá para entrenar el árbol b.

El proceso de entrenamiento consiste en iterar sobre $b = 1, ..., B$, entrenando en cada iteración el árbol de decisión f_b con el conjunto T_b. Luego, la predicción del modelo completo se puede escribir como:

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

Este cálculo representa el promedio de las decisiones de cada árbol ante la entrada x', esto funciona bien en casos donde las clases posibles para un elemento pertenecen a un espacio continuo. En caso de tener un número limitado de clases en un espacio discreto, se utiliza votación mayoritaria entre todos los arboles para obtener la predicción del modelo completo.

2.4.3. K Nearest Neighbors

K Nearest Neighbors (KNN) [10] es un método de clasificación que basa su funcionamiento en estimar la función de densidad de probabilidad de que un elemento dado pertenezca a una clase C. De esta forma se establece un valor k, el cual dependerá de los datos y deberá ser entrenado. Este valor k permite obtener, usando una función de distancia, los k elementos más cercanos considerando espacio de características, finalmente la predicción corresponderá a la clase predominante dentro de los k elementos obtenidos.

Sea $x = (c_1, c_2, ..., c_n)$ un elemento en un espacio de características X y sea $d(x_i, x_j)$ una función que determina la distancia en X entre dos elementos. Obtener la clase correspondiente al clasificar un elemento x consiste en obtener los k elementos mas cercanos en X. Sean $x_1, x_2, ..., x_k$ los elementos más cercanos a x entonces la clase asignada por el modelo clasificador sera la mas frecuente dentro de $x_1, x_2, ..., x_k$.

2.4.4. Regresión Logística

Regresión Logística (*Logistic Regression*) [11] es un método de regresión el cual se diferencia de los métodos de clasificación vistos en que el resultado predicho no pertenece a un conjunto de clases establecido y limitado, sino que puede ser cualquier valor perteneciente a una escala continua, generalmente en los números reales.

Este método se puede usar para solucionar el problema de la clasificación binaria, para ello se hace uso de una función logística para modelar la probabilidad de que un elemento x, de tamaño p, pertenezca a una clase. Con ello se define:

$$z = b_0 + b_1 x_1 + ... + b_p x_p$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

donde los valores b_i son parámetros entrenables. La función f es una función *sigmoid* la cual tiende rápidamente a 0 o 1 con valores de z altos o pequeños respectivamente. Generalmente, la clasificación, en el conjunto de clases $\{0, 1\}$, se logra aproximando los valores de f de forma que cuando son mayores a 0.5 se transforman en 1 y el resto se transforman en 0.

El entrenamiento de los valores b_i se basa en minimizar la función de costo definida como:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\left(y^i \log\left(\hat{y}^i\right) + (1-y^i)\log\left(1-\hat{y}^i\right)\right)$$

donde y^i es la clase real y $\hat{y}^i$ la clase estimada. Con esta ecuación el valor de cada b_j en cada ciclo de entrenamiento viene dado por:

$$b_j = b_j - \alpha\sum_{i=1}^{N}(\hat{y}^i - y^i)x_j^i$$

Como se indicó previamente, este modelo permite resolver el problema de la clasificación binaria, en el caso de la clasificación multiclase, generalmente se crea un modelo por cada clase y se sigue la estrategia de *one vs rest*. A cada modelo se le asigna una clase y este predice si una entrada pertenece a su clase asignada o no.

2.4.5. Gaussian Naive Bayes

Gaussian Naive Bayes [12] es un método de clasificación que basa su funcionamiento en la premisa de que cada una de las características del vector de características son independientes entre sí. De esta forma, usando el teorema de Bayes, se calcula probabilidad de que un vector de características $(x_1, ...,_{x\,n})$ pertenezca a una clase y.

$$P(y \mid x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n \mid y)}{P(x_1, ..., x_n)}$$

dado la definición de probabilidad condicional, la formula se puede reescribir como:

$$P(y \mid x_1, ..., x_n) = \frac{P\,(y)\ \prod_{i=1}^{n}\ P\,(x_i \mid y)}{P(x_1, ..., x_n)}$$

el termino del denominador es una constante que solo depende del x de entrada, por ende es posible abstraerse de su valor:

$$P(y \mid x_1, ..., x_n) = \frac{1}{Z}\,P\,(y)\ \prod_{i=1}^{n}\ P\,(x_i \mid y)$$

con ello, es posible clasificar obteniendo el valor y' para el cual $P(y' \mid x_1, ..., x_n)$ es máximo. Es decir:

$$\hat{y} = arg\max_{y}\ P\,(y)\ \prod_{i=1}^{n}\ P\,(x_i \mid y)$$

El calculo de $P(x_i \mid y)$ para cada $i \in 1, ..., n$ asume que la probabilidad de cada una de las características x_i sigue un distribución gaussiana, por lo cual se puede calcular como:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}}e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$

donde σ_y y μ_y representan de la varianza y el promedio de la clase y respectivamente.

2.4.6. Perceptrón Multicapa

Perceptrón Multicapa [5] es un método de clasificacion, el cual basa su funcionamiento en el uso de múltiples perceptrones individuales unidos entre sí en un sistema de capas. Un perceptrón es un modelo simple de clasificacion binaria, el cual funciona con una función de activación, la cual para ciertos valores de entrada entrega como resultado 1 y para el resto de valores entrega 0. La forma más simple de perceptrón y que es muy común en el estado del arte, es aquel en el cual los valores positivos son aproximados a 1 y los valores negativos a 0, con esto un perceptrón se puede representar como:

$$y_i = \varphi(w_i x_i) = \begin{cases} 1 & w_i x_i \geq 0 \\ 0 & w_i x_i < 0 \end{cases}$$

donde w_i es un parámetro entrenable, conocido como peso sináptico y que permite determinar la aportación al resultado final del elemento x_i.

En el caso de perceptrones multicapa, los perceptrones son divididos en múltiples capas. En cada capa cada perceptron trabaja de forma independiente al resto de perceptrones, pero sus valores de entrada dependen de lo que entregue la capa anterior (ver Figura 2.6).

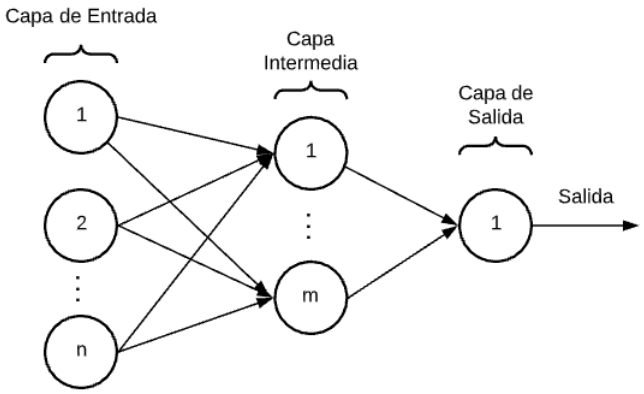


Figura 2.6: Arquitectura de un Perceptrón Multicapa

La estructura mostrada presenta de forma simple a un Perceptrón Multicapa. La capa de entrada recibe el vector de características desde el dataset, la o las capas intermedias procesan, de forma secuencial, lo que reciben de su capa predecesora para finalmente entregar un vector de características procesado a la capa de salida, la cual se encargará de realizar la predicción del modelo de clasificación.

Para lograr la clasificación el Perceptrón Multicapa realiza dos procesos esenciales:

- Propagación: Es el proceso que sigue el clasificador para, a través de cada una de las capas, procesar los datos de entrada y para entregar una predicción.
- Aprendizaje (*backpropagation*): luego de realizar la propagación, se realiza un cálculo del error al clasificar, este error se propaga desde la capa de salida hacia la capa de entrada y permite modificar los pesos sinápticos, buscando mejorar la predicción del modelo.

2.5. Ensemble de Clasificadores

Los ensembles de clasificadores [13] son métodos de machine learning que permiten usar varios modelos clasificadores de forma conjunta, con el objetivo de obtener mejores resultados que los obtenidos de forma individual.

Los ensembles se suelen clasificar en cuatro tipos:

- **Stacking**
- **Voting**
- **Bagging**
- **Boosting**

2.5.1. Stacking

El método de ensemble *Stacking* [14] permite el uso en paralelo de varios clasificadores independientes, de esta forma permite agrupar las predicciones de cada uno de estos clasificadores formando un nuevo vector de características que funcionará como datos de entrada para un nuevo modelo clasificador.

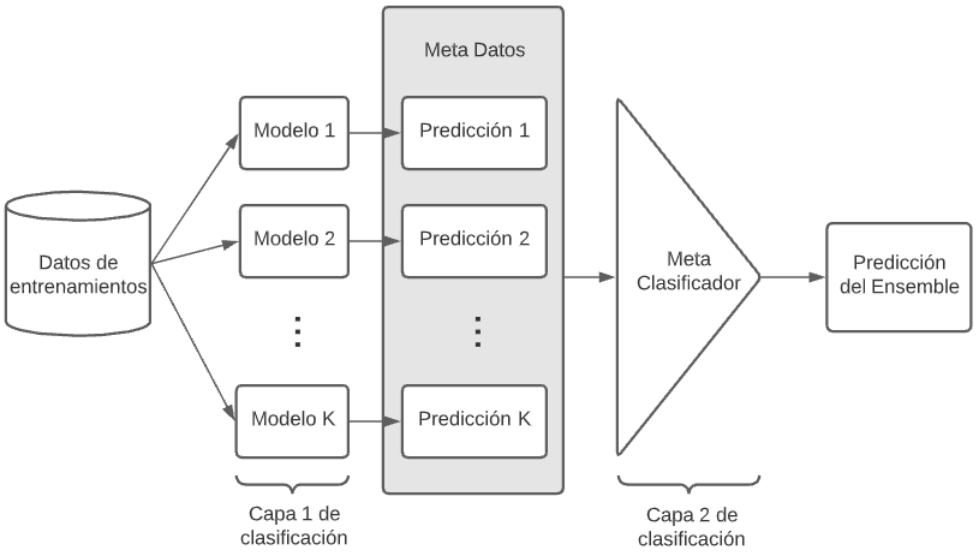


Figura 2.7: Arquitectura de un ensemble de tipo *Stacking*

Como se muestra en la Figura 2.7, este tipo de ensemble genera dos capas de clasificación, la primera capa está conformada por el conjunto de los clasificadores clásicos, los cuales al procesar el dataset generan, en conjunto, nuevas instancias de datos que usualmente son referidas, en el estado del arte, como *meta-instances* o *meta-data* (meta-datos) [15]. La segunda capa de clasificación es la que procesa los meta-datos generados, tanto para entrenar como para realizar una predicción, en esta segunda capa se ubica un nuevo clasificador que debido a su condición de pertenecer a una nueva capa de clasificación se le da el nombre de *meta-classifier* o *meta-learner* (meta clasificador) [15], este *meta-learner* debe estar correctamente

construido, tanto como para aceptar la estructura de los meta-datos como para entregar una predicción en el formato requerido.

2.5.2. Voting

Los ensembles de *voting* son muy similares a los basados en el método de *stacking* previamente visto, la diferencia principal es que en este caso no se usa un meta-learner, sino que, como su nombre en español indica, este tipo de ensembles se basa en votación. Los modelos clasificadores en la primera capa clasificadora reciben la misma información (conjunto de entrenamiento), con esto se entrenan y luego cada uno entrega, de forma independiente, una predicción acerca de la clase a la que pertenece una entrada x a clasificar, tomando la predicción de cada modelo se realiza una votación donde la clase con más votos se convierte en la predicción final del ensemble.

Para la fase de votación se puede tener en cuenta que la votación de cada modelo tiene el mismo peso que la de los demás o se puede asignar un peso a cada una de las predicciones, siendo estos pesos directamente proporcionales a la precisión individual del modelo que emitió la predicción.

2.5.3. Bagging

El método de ensemble *bagging* [16] funciona de forma similar a los dos descritos previamente, se busca variedad en los modelos clasificadores y con ello una variedad en las predicciones que permitan un uso más efectivo de las respuestas combinadas. A diferencia de *Voting* y *Stacking*, en este tipo de ensembles, la variabilidad en los modelos clasificadores no se obtiene al usar distintos tipos o construirlos con diferentes estructuras, por el contrario, se usan clasificadores iguales o similares, pero el entrenamiento de estos no se realiza con los mismos datos, se generan nuevos conjuntos de entrenamiento para cada clasificador, logrando variabilidad a partir del entrenamiento. La Figura 2.8 muestra la estructura general de este tipo de ensemble.

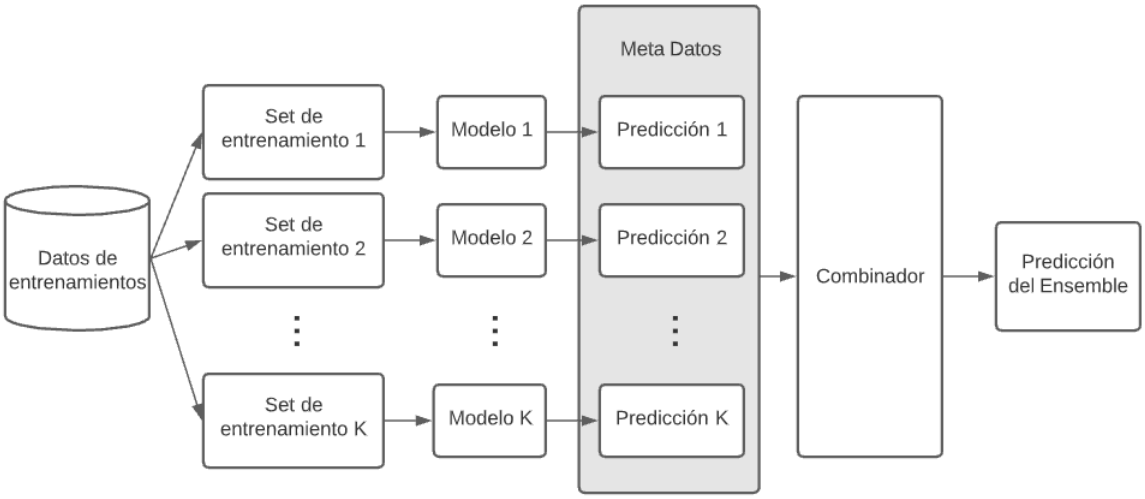


Figura 2.8: Arquitectura de un ensemble de tipo *Bagging*

Debido a que se requiere una buena variabilidad en el entrenamiento de los modelos que en un principio son iguales o similares, es necesario que los sets de entrenamiento de cada clasificador sean lo suficientemente distintos entre sí, esto puede provocar que un ensemble de este tipo y que use un gran número de clasificadores requiera de un dataset lo suficientemente grande para generar set de entrenamientos de buena calidad para mantener la variabilidad de los modelos.

2.5.4. Boosting

El método de ensemble *boosting* [17] se diferencia de los métodos descritos previamente en que las predicciones de cada uno de los modelos que conforma el *ensemble* no son independientes unas de otras, en este método los clasificadores funcionan en secuencia, de forma que la predicción de un clasificador se convierte en la entrada de otro, con esto se espera que la respuesta vaya mejorando su precisión conforme pasa por cada uno de los clasificadores. La Figura 2.9 muestra la estructura general de este tipo de ensemble.

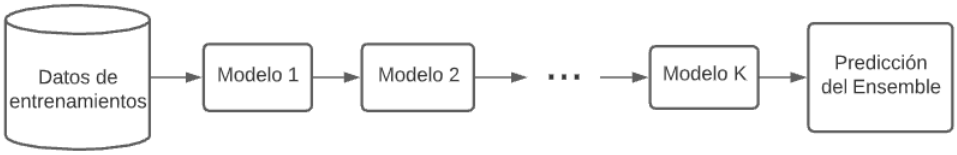


Figura 2.9: Arquitectura de un ensemble de tipo *Boosting*

2.6. Métodos de Ensemble

Como se indicó anteriormente, usualmente los ensembles se pueden clasificar en cuatro tipos: *stacking*, *voting*, *bagging* y *boosting*. Con esta clasificación y con su forma de funcionamiento es posible intuir que no es posible aplicar los cuatro tipos descritos en la solución del problema planteado en este trabajo.

Es fácil notar que los modelos descriptor-clasificador con los que se cuentan no están capacitados para implementar un *ensemble* de tipo *boosting*, dado que sería necesario que los distintos modelos se pudieran conectar de forma secuencial, es decir, que el resultado de salida de algunos sea del mismo formato que la entrada de otros, esto es algo que no se puede realizar sin cambiar los modelos ya implementados, lo cual va más allá de los objetivos planteados para esta memoria. De forma similar ocurre con los *ensembles* de tipo *bagging*, donde usualmente se usan modelos iguales, pero con distintos datos de entrenamiento, en este caso todas las combinaciones descriptor-clasificador son muy distintas tanto en su construcción como en sus resultados, además, la limitación en el tamaño del conjunto de datos con el que se cuenta no permitiría realizar *bagging* con demasiados modelos a la vez.

De lo anterior, los métodos de ensembles que se ajustan al objetivo de este trabajo de memoria son los pertenecientes a los tipos *voting* o *stacking*.

Una característica que se debe considerar de estos tipos de ensembles es que presentan una primera capa en la cual se tiene una cantidad definida de modelos de clasificación, además de una capa de meta-clasificación, esto genera que la complejidad del ensemble sea del orden de la suma de las complejidades de los modelos individuales sumado con la complejidad del meta-clasificador. En los casos donde una misma instancia de un modelo se utiliza múltiples veces solo se contabiliza una vez para el cálculo de la complejidad. De esta forma, el costo de usar ensembles, dependiendo de los clasificadores en la primera capa, es varias veces mayor a usar modelos individuales y puede resultar en un punto determinante para decidir si usar ensembles o no.

2.6.1. Plurality Voting

Plurality Voting (Votación Mayoritaria) [18] es un método de *ensemble* perteneciente a los de tipo *voting*, el cual plantea un sistema de votación mayoritario simple, es decir, las predicciones de cada uno de los modelos tienen igual peso en la predicción final del *ensemble*.

Sea $(x_i, y_i) \in T$ pares del conjunto de entrenamiento y sea $H^{'} = \{h_1, ..., h_L\}$ con $H^{'} \subseteq H$ un subconjunto del conjunto de todos los modelos descriptor-clasificador. Con esto la clasificación por *Plurality Voting* se puede escribir como:

$$y = maxRepValue([h_1(x_i), ..., h_L(x_i)])$$

donde $[h_1(x_i), ..., h_L(x_i)]$ es un arreglo que guarda la predicción de cada uno de los modelos de $H^{'}$ y $maxRepValue$ es una función que retorna el valor que se repite más veces en un arreglo, eligiendo al azar entre los valores con mayor cantidad de repeticiones en caso de empate en la mayoría. En la Figura 2.10 se muestra la estructura de este modelo, y de forma análoga a como se observa en la Figura 2.7, existe una segunda capa de clasificacion, pero en este caso no se utiliza un meta clasificador, sino que se hace uso de la función $maxRepValue$ para realizar la votación mayoritaria.

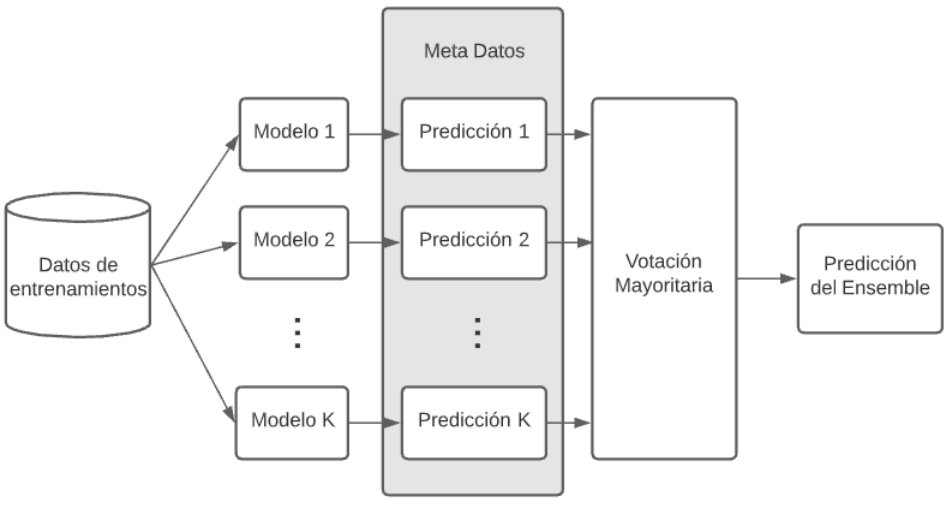


Figura 2.10: Arquitectura del ensemble *Plurality Voting*

2.6.2. Weighted Plurality Voting

Weighted Plurality Voting (Votación Mayoritaria con Peso) es un método de *ensemble* perteneciente a los de tipo *voting*, el cual se diferencia de *Plurality Voting* en que plantea un sistema de votación con peso, donde la predicción de cada modelo obtiene un valor proporcional a su precisión individual. Se puede expresar como:

$$y = mostVotedValue([(h_1(x_i), w_1), ..., (h_L(x_i), w_L)])$$

donde los $w_1, ..., w_L$ representan los pesos asignados a cada uno de los modelos h_i, *mostVotedValue* es una función la cual, para cada valor y_k, suma los pesos correspondientes a los modelos que cumplen que $h_i(x_j) = y_k$. De esta forma, el valor entregado por el ensemble es aquel y_k cuya suma de pesos sea la mayor. Como se muestra en la Figura 2.11, la estructura es similar a *Plurality Voting*, existe una segunda capa de clasificación donde se hace uso de la función *mostVotedValue* para realizar el proceso de votación mayoritaria con peso.

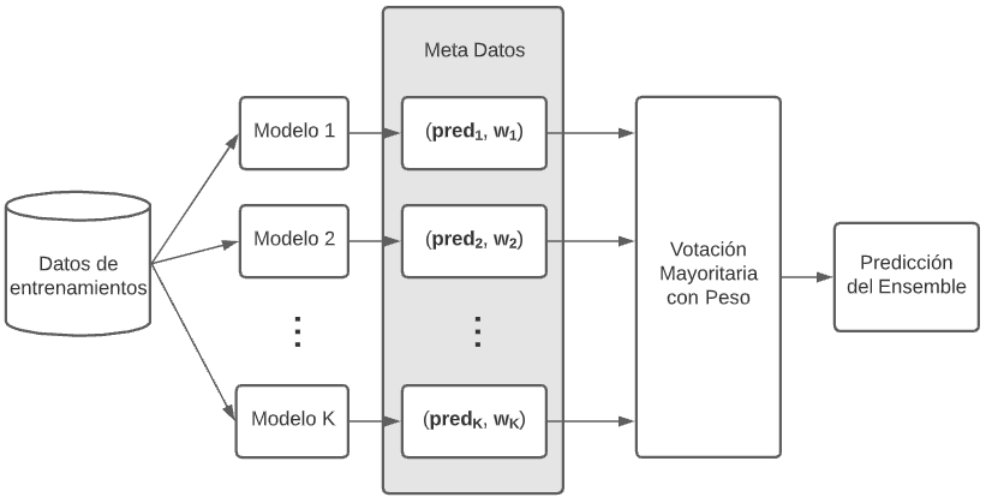


Figura 2.11: Arquitectura del ensemble *Weighted Plurality Voting*

2.6.3. Stacked Ensemble

Stacked Ensemble es un método de *ensemble* perteneciente a los de clase *stacking*, su funcionamiento es lo más básico en los métodos de *stacking*, se toma un conjunto de modelos de clasificación, cada uno de los clasificadores pertenecientes al conjunto realiza su predicción de forma independiente, los resultados de estas predicciones se entregan como entrada de datos a un meta-clasificador el cual entregara la predicción final del ensemble.

Sea $(x_i, y_i) \in T$ pares del conjunto de entrenamiento, $H^{'} = \{h_1, ..., h_L\}$ con $H^{'} \subseteq H$ un subconjunto del conjunto de todos los modelos descriptor-clasificador y $h^{'}$ un meta-clasificador de aprendizaje supervisado con tamaño de entrada igual a L. La clasificación usando Stacked Ensemble se puede escribir como:

$$y = h^{'}([h_1(x_i), ..., h_L(x_i)])$$

2.6.4. One vs Rest Ensemble

One vs Rest Ensemble (basado en *one-class classifier ensemble* [19]) es un método de *ensemble* perteneciente a los de tipo *stacking*, su funcionamiento se basa en formar N grupos de K modelos descriptor-clasificador distintos, donde N es la cantidad de clases posibles para clasificar y K es un número natural menor a la cantidad total de modelos descriptor-clasificador. A cada uno de los grupos formados se les asigna una clase, con la cual los modelos pertenecientes al grupo deben entregar 1 si una entrada de datos dada pertenece a la clase asignada o 0 en caso contrario. Los valores entregados por todos los grupos se juntan y se entregan a un meta-clasificador supervisado el cual termina el proceso de clasificación retornando una predicción de la clase (ver Figura 2.12).

Sea $(x_i, y_i) \in T$ pares del conjunto de entrenamiento, $\Omega = \{0, ..., N-1\}$ clases posibles para realizar la predicción, $H_i = \{h_1^{(i)}, ..., h_K^{(i)}\}$ con $i = \{0, ..., N-1\}$, $H_i \subseteq H$ subconjunto del conjunto de todos los modelos descriptor-clasificador y $h^{'}$ un meta-clasificador de aprendizaje supervisado con tamaño de entrada igual a $N \cdot K$. La clasificación usando One vs Rest Ensemble se puede escribir como:

$$y = h^{'}([I(h_1^{(0)}(x_i) = 0), ..., I(h_K^{(0)}(x_i) = 0), ..., I(h_1^{(N-1)}(x_i) = N-1), ..., I(h_K^{(N-1)}(x_i) = N-1)])$$

donde la función $I(h_i^{(j)}(x) = \omega)$ es una función indicadora que entrega 1 si $h_i^{(j)}(x)$ es igual a ω o 0 en caso contrario.

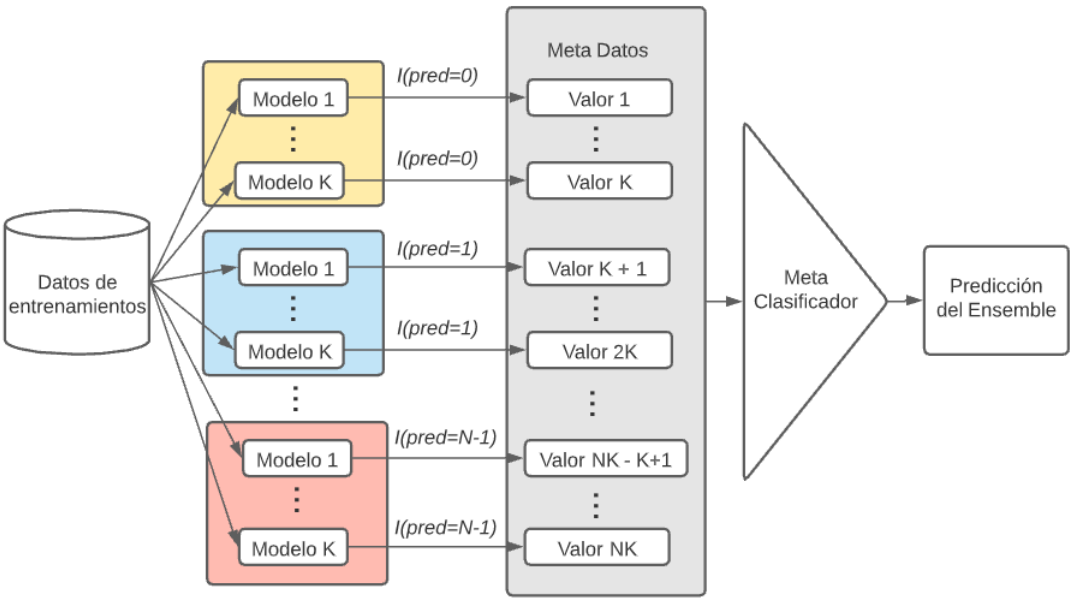


Figura 2.12: Arquitectura de *One vs Rest Ensemble*

2.6.5. Super Learner

Super Learner [20] es un método de *ensemble* perteneciente a los de tipo *stacking*, su funcionamiento es similar al método *Weighted Plurality Voting* (ver Capítulo 2.6.2), se calcula el

grado de aportación al resultado final de cada uno de los modelos pertenecientes al ensemble, este grado de aportación se obtiene al entrenar y probar cada modelo con distintos conjuntos de datos, para ello se utiliza el método de validación cruzada (*cross validation*) para separar el conjunto de datos en partes más pequeñas, se utilizan todas estas partes menos una (se deja esta última para la validación) para obtener un riesgo asociado a cada modelo, este riesgo representa la magnitud de los errores del modelo, por ende el peso asignado a cada clasificador debe ser inversamente proporcional a la magnitud del riesgo.

Teniendo los pesos, los resultados se calculan a través de una votación mayoritaria con peso, convirtiendo a este tipo de ensemble una variación entrenable de *Weighted Plurality Voting*.

Capítulo 3

Metodología experimental

En este capítulo se estudiará la metodología usada en el desarrollo de este trabajo, además de revisar la forma en que se usaran las distintas herramientas disponibles en el estado del arte. También, se presentará el diseño de los experimentos que se realizarán.

En este trabajo de memoria la metodología se centrará en la implementación

3.1. Dataset

El dataset que se utiliza en este trabajo fue generado por la Facultad de Comunicaciones de la Pontificia Universidad Católica para el Instituto Milenio Fundamentos de los Datos (IMFD). Se cuenta con un set inicial de 618.284 imágenes recolectadas de mensajes de usuarios chilenos publicados en twitter entre mayo y octubre de 2019, de este set inicial se extrajo una muestra de 52000 imágenes que fueron clasificadas manualmente por expertos.

3.1.1. Medidas de fiabilidad

Inter-coder reliability (ICR)

ICR [21, 22] es una medida numérica que permite determinar cuan en acuerdo se encuentran dos o más codificadores distintos con respecto a como una misma información debe ser codificada, cumpliéndose que todos los codificadores manejan el mismo esquema de codificación. En este caso de estudio, el ICR se entiende como el porcentaje de imágenes en la que la clasificación dada por todos los expertos concuerdan.

Alfa de Krippendorf

El Alfa de Krippendorf es una medida de fiabilidad que, al igual que ICR, permite determinar el nivel de acuerdo entre dos o más codificadores. Se define como:

$$\alpha = 1 - \frac{D_o}{D_e}$$

donde D_o representa el nivel de desacuerdo obtenido entre los observadores, y D_e el desacuerdo esperado. De esta forma, un nivel de desacuerdo D_o cercano a 0, implica un valor Alfa de Krippendorf cercano a 1, representando una fiabilidad máxima entre los observadores.

Por otro lado, cuando $D_o = D_e$ el desacuerdo es el máximo esperado entre los observadores, resultando en una fiabilidad nula.

3.1.2. Metodología de anotación

Con las medidas de fiabilidad definidas, se entrenaron expertos de modo que se pudiera alcanzar un ICR del 90 % y un Alfa de Krippendorf de 0.7 o superior. Obteniendo estos cálculos con una muestra de 2000 imágenes extraídas del conjunto de datos de 52000 imágenes.

Luego de obtener la fiabilidad deseada, cada uno de los expertos clasificó 13000 imágenes distintas en 4 clases: Meme, No-Meme, Sticker y Dudoso. Esta clasificación toma en cuenta todas las características presentes en las imágenes, incluyendo el contenido textual. Al usar esta metodología de anotación en el dataset de 52000 imágenes, se obtuvo que 1194 imágenes corresponden a Meme, 1443 a Sticker, 49347 a No-Meme y 16 a Dudoso.

3.2. Métricas

En este tipo de trabajos, en los cuales se desarrollan varios métodos que buscan solucionar un problema, es importante definir ciertas métricas que permitan declarar de forma objetiva que modelo resuelve mejor el problema.

En el caso del problema de la clasificación, las métricas permiten establecer el desempeño, ya sea de clasificadores individuales o *ensembles* al entrenarse y ponerse a prueba con un determinado conjunto de datos. El uso de métricas generaliza los resultados permitiendo predecir el comportamiento del modelo ante nuevos datos a evaluar, lo cual es útil, entre otras cosas para prever el sobreajuste (*Overfitting*) [23] de los modelos. De esta forma, las métricas usadas en este trabajo concuerdan con las más usadas en el estado del arte, las cuales son: Precisión, Exactitud (*Accuracy*), Exhaustividad (*Recall*) y Puntuación F1 (*F1 Score*).

Con el fin de realizar el cálculo de las métricas mencionadas, es necesario definir una matriz de confusión 3.1, la cual permite visualizar el desempeño de un clasificador.

		Clase predicha	
		1	0
Clase real	1	Verdadero Positivo (VP)	Falso Negativo (FN)
	0	Falso Positivo (FP)	Verdadero Negativo (VN)

Tabla 3.1: Matriz de confusión

3.2.1. Exactitud (Accuracy)

La exactitud es la proporción de casos evaluados correctamente (ya sea positivos o negativos) por sobre todos los casos evaluados. Se expresa como:

$$Exactitud = \frac{VP + VN}{VP + FP + FN + VN}$$

Su valor va entre 0 y 1. En el caso de la clasificación multiclase, se puede obtener como

el promedio de las exactitudes de cada una de las clases, calculadas individualmente como indica la fórmula, o bien, tomando simplemente el total de aciertos dividido por la cantidad total de ejemplos.

3.2.2. Precisión

La precisión es la proporción de casos evaluados correctamente como positivos por sobre todos los casos evaluados como positivos. Se expresa como:

$$Precisión = \frac{VP}{VP + FP}$$

Su valor va entre 0 y 1. En el caso de la clasificación multiclase, se puede obtener como el promedio de las precisiones de cada una de las clases, calculadas individualmente como indica la fórmula.

3.2.3. Exhaustividad (Recall)

La exhaustividad es la proporción de casos evaluados correctamente como positivos por sobre el total de casos realmente positivos. Se expresa como:

$$Exhaustividad = \frac{VP}{VP + FN}$$

Su valor va entre 0 y 1. En el caso de la clasificación multiclase, se puede obtener como el promedio de las exhaustividades de cada una de las clases, calculadas individualmente como indica la fórmula.

3.2.4. Puntuación F1 (F1 score)

La puntuación F1 es el promedio armónico entre la precisión y la exhaustividad. Se expresa como:

$$F1 = 2 \cdot \frac{Precisión \cdot Exhaustividad}{Precisión + Exhaustividad} = \frac{VP}{VP + \frac{1}{2}(FN + FP)}$$

Su valor va entre 0 y 1. En el caso de la clasificación multiclase, se puede obtener como el promedio de las puntuaciones F1 de cada una de las clases, calculadas individualmente como indica la fórmula.

3.3. Modelos descriptor-clasificador

Para cumplir con el objetivo de este trabajo de memoria se hace uso de varios modelos descriptores de imagen y clasificadores presentes en el estado del arte, específicamente en el trabajo de memoria "Comparación de descriptores para clasificación de memes" de Alberto Sara (2020) [1], los cuales fueron descritos en el Capítulo 2.1 y que se resumen en la siguiente tabla.

Descriptores	Clasificadores
Distribución de Color	Clasificador de Vectores de Soporte (SVC)
Histograma de Grises	Random Forest
Histograma de Bordes	K Nearest Neighbors
Histograma de Gradientes Orientados	Regresión Logística
DeCAF7	Gaussian Naive Bayes
ResNet152	Perceptrón Multicapa

Tabla 3.2: Resumen de los descriptores y clasificadores a utilizar

De acuerdo con los combinadores de clasificadores (*Ensembles*) que se usarán en este trabajo (ver Capítulo 2.6) la forma de utilizar los modelos descriptor-clasificador sigue la línea de lo establecido para métodos de ensembles *voting* y *stacking* para crear una primera capa de clasificación, donde cada uno de los modelos es entrenado de forma independiente con el mismo dataset. De esta forma, cada uno de los ensembles usará o bien todos los modelos descriptor-clasificador del estado del arte o tan solo un subconjunto de ellos.

Como se puede observar por la clasificación hecha por expertos, de las 52000 imágenes en el dataset, descartando la clase Dudoso, existe un gran desbalance en la cantidad de ejemplos disponibles por clase, sobre todo entre No-Meme y el resto. Esto puede resultar en un problema al momento de entrenar los modelos, pudiéndose producir el ya mencionado sobreajuste (*Overfitting*). Para evitar esto se plantean dos métodos de entrenamiento, *Undersampling* y *Cost-sensitive*, con cada uno de estos métodos de entrenamiento se prueban los ensembles de forma independiente para obtener resultados separados.

Entrenamiento Undersampling

El primer método de entrenamiento es el mismo usado en el estado del arte [1], teniendo un dataset con un desbalance en los datos, en este caso con la clase No-Meme siendo la predominante, el entrenamiento con Undersampling reduce el dataset eliminando elementos de las clases mayoritarias con el objetivo de obtener un conjunto de datos donde cada una de las clases tiene la misma cantidad de ejemplos. Luego de obtener un conjunto de datos equilibrado, se realiza validación cruzada (*Cross-Validation*) para evaluar los resultados de los distintos modelos.

Tomando como ejemplo el dataset de imágenes disponible, este método de entrenamiento reduciría la cantidad de ejemplos a usar de las clases No-Meme y Sticker para equiparar con los disponibles de la clase Meme, resultando en un nuevo dataset donde cada una de las clases contaría 1194 (cantidad de imágenes clasificadas como Meme) ejemplos.

Entrenamiento Cost-sensitive

Aunque el método de entrenamiento *Undersampling* es muy útil, debido a su facilidad para implementar y para analizar los resultados obtenidos a través de las métricas, tiene el inconveniente de que no aprovecha del todo el dataset disponible. Para solventar esto, se plantea el método de entrenamiento *Cost-sensitive*, el cual utiliza el dataset completo, haciendo uso de cada uno de los ejemplos disponibles. Sin embargo, entrenar cada modelo con el dataset completo, siendo este tan desbalanceado, ocasionará *Overfitting*, esto se ve más

claro al tomar en cuenta que un clasificador que solo entrega como predicción la clase con mayor cantidad de ejemplos (No-Meme) puede lograr una exactitud de más del 90 %. De esta forma, en este método de entrenamiento la exactitud pierde relevancia dándole mayor énfasis a las otras métricas de evaluación.

Cost-sensitive basa su funcionamiento en la premisa de que no todos los errores tienen la misma importancia. De esta forma, los errores de predicción en las clases con menos ejemplos pesan más que los ocurridos con las clases mayoritarias, esto se ve reflejado en un mayor cambio en los parámetros del clasificador al momento de entrenarlo.

Al igual que en el entrenamiento *Undersampling*, en este caso también se acompaña el uso de esta técnica con validación cruzada para evaluar los resultados de los modelos.

3.4. Experimentación

Al enfocarse en distintas métricas, los dos métodos presentados para solucionar el problema de las clases desbalanceadas deben analizarse de forma separada. Por ello, la metodología experimental de este trabajo separa la presentación de los resultados en dos experimentos:

- **Experimento Undersampling:** Evaluación de desempeño cada uno de los métodos propuestos de ensemble, utilizando el método *undersampling* para preprocesar el dataset
- **Experimento Cost-sensitive:** Evaluación de desempeño de los modelos descriptor-clasificador de forma individual y de la combinación de ellos a través de los ensembles propuesto, utilizando el método de entrenamiento *Cost-sensitive*.

3.4.1. Experimento Undersampling

Este experimento evalúa el desempeño de los distintos métodos de ensembles propuestos en el Capítulo 2.6 utilizando la técnica *undersampling*. Por cada uno de los ensembles se obtienen tres resultados para las distintas métricas, estos resultados dependen del subconjunto de combinaciones descriptor-clasificador con las que el ensemble es configurado. Para ello se establecen los tres subconjuntos de modelos descriptor-clasificador:

- **Subconjunto 1:** este subconjunto utiliza los 36 modelos resultantes de combinar los 6 descriptores con cada uno de los 6 clasificadores disponibles (ver Tabla 3.2).
- **Subconjunto 2:** este subconjunto utiliza 18 modelos, resultantes de combinar los 6 clasificadores disponibles con los 3 descriptores con mejor desempeño en los resultados individuales: Histograma de Gradientes Orientados, DeCAF7 y ResNet152.
- **Subconjunto 3:** este subconjunto utiliza 22 modelos, resultantes de añadir al subconjunto 2, las combinaciones: Histograma de Grises con *K Nearest Neighbors* y *Random Forest*, Histograma de Bordes con SVC y Distribución de Color con *K Nearest Neighbors*. Estos últimos cuatro modelos que se agregan, son aquellos que, a través de un análisis de desempeño por imagen, se determinó que tenían mejores resultados en las imágenes mas difícil de clasificar, osea, aquellas que son clasificadas correctamente por solo unos pocos modelos descriptor-clasificador.

3.4.2. Experimento Cost-sensitive

Este experimento evalúa el desempeño, tanto de los clasificadores individuales como de los distintos métodos de ensembles utilizando la técnica de entrenamiento *Cost-sensitive*. Al usar este tipo de técnica, los modelos descriptor-clasificador se ven limitados a aquellos que utilizan clasificador *Random Forest*, *Logistic Regression* o SVC, dado que estos son los modelos que permiten usar *Cost-sensitive* con la librería *scikit-learn* [24] usada en este trabajo de memoria. De esta forma, en este experimento solo se tiene en consideración el conjunto de modelos definidos por la combinación de los 3 clasificadores mencionados y los 6 descriptores totales. Para el correcto entrenamiento de los clasificadores se definen los pesos de cada una de las clases como una ponderación aproximada entre los ejemplos de la clase sobre el total de imágenes en el dataset, resultando en los siguientes valores:

$$NoMeme : 0.012 \qquad Meme : 0.6 \qquad Sticker : 0.43$$

Debido a que no se tienen resultados previos, para el dataset del cual se dispone, que utilicen *Cost-sensitive*, en este experimento, además de probar el desempeño de los ensembles también se prueba el rendimiento de los modelos descriptor-clasificador de forma individual. Sumado a esto, se obtienen las métricas resultantes para un clasificador que solo elige la clase con mayor cantidad de ejemplos (No-Meme), permitiendo realizar un análisis más profundo tanto de los resultados individuales como de los provistos por los ensembles.

3.5. Configuración de Meta clasificadores

La implementación de ensembles de tipo *Stacking* (ver Capítulo 2.5.1), implica el uso de meta-clasificadores para procesar los meta-datos generados por los modelos individuales. Por ello, es necesario construir nuevos clasificadores que se adapten a los ensembles que lo requieran. De esta forma, en este trabajo se usarán como meta-clasificadores los mismos clasificadores descritos en la sección 2.4 y que se usan en combinación con los descriptores para la generación de los meta-datos.

Dado que los meta-clasificadores reciben distintos datos que los modelos de clasificacion individuales, es necesario obtener sus propios hiperparámetros optimizados, en caso de que lo requieran. Para la obtención de los hiperparámetros de un meta clasificador específico para un cierto método de ensemble, primero se define un rango de hiperparámetros finitos, luego se entrena una cantidad dada de veces con cada combinación de parámetros, finalmente se obtienen las métricas usando los datos de prueba y se entrega como mejor combinación la que diera lo mejores resultados en las métricas.

Para ello se toma en cuenta el espacio de hiperparámetros presentado en la Tabla 3.3, los cuales toman en cuenta valores que se adaptan al problema presentado.

Clasificador	**Espacio de Parámetros**	
Perceptrón Multicapa	Capas ocultas (hidden layer)	(3000), (500, 10), (500)
	Función de activación	tanh, relu
	Learning Rate	adaptive, invscaling
	Solver	sgd, adam
	Alpha	0.0001
KNN	Algoritmo	kd_tree, ball_tree
	Número de vecinos	3, 12, 20
	Tipo de distancia	2, 3, 7
SVC	Kernel	rbf, poly
	Parámetro de regularización	0.1
	Grado	1, 3, 5
	Coeficiente	0, 1

Tabla 3.3: Espacio de Hiperparámetros

La existencia de dos experimentos a evaluar, obliga a calcular hiperparámetros distintos para los mismos ensembles en distintos experimentos, los resultados de estos cálculos se pueden observar en las siguientes tablas.

Clasificador	Parámetros	
Perceptrón Multicapa	Capas ocultas (hidden layer)	(3000)
	Función de activación	tanh
	Learning Rate	adaptive
	Alpha	0.0001
	Solver	adam
KNN	Algoritmo	kd_tree
	Número de vecinos	3
	Tipo de distancia	2
SVC	Kernel	rbf
	Parámetro de regularización	0.1
	Grado	1
	Coeficiente	0

Tabla 3.4: Hiperparámetros optimizados para el método de ensemble *Stacked* en el experimento 1

Clasificador	Parámetros	
Perceptrón Multicapa	Capas ocultas (hidden layer)	(3000)
	Función de activación	tanh
	Learning Rate	adaptive
	Alpha	0.0001
	Solver	adam
KNN	Algoritmo	ball_tree
	Número de vecinos	12
	Tipo de distancia	2
SVC	Kernel	poly
	Parámetro de regularización	0.1
	Grado	1
	Coeficiente	0

Tabla 3.5: Hiperparámetros optimizados para el método de ensemble *One vs Rest* en el experimento 1

Clasificador	Parámetros	
SVC	Kernel	rbf
	Parámetro de regularización	0.1
	Grado	1
	Coeficiente	0
	Pesos de Clases	{**0:** 0.012, **1:** 0.6, **2:** 0.43}

Tabla 3.6: Hiperparámetros optimizados para el método de ensemble *Stacked* en el experimento 2

Clasificador	Parámetros	
SVC	Kernel	rbf
	Parámetro de regularización	0.1
	Grado	1
	Coeficiente	0
	Pesos de Clases	{**0:** 0.012, **1:** 0.6, **2:** 0.43}

Tabla 3.7: Hiperparámetros optimizados para el método de ensemble *One vs Rest* en el experimento 2

Capítulo 4

Resultados Experimentales

En este capítulo se presentan los resultados obtenidos a través de cada uno de los experimentos realizados, incluyendo un análisis de estos y comparándolos con los resultados presentes en el estado del arte.

4.1. Resultados Previos

Con el fin de permitir un mayor entendimiento de las comparaciones y análisis que se realizaran en este capítulo, en esta sección se presentan los resultados previos, correspondientes a lo obtenido en el trabajo de memoria "Comparación de descriptores para clasificación de memes" de Alberto Sara (2020) [1]. Estos resultados miden el desempeño de cada una de las combinaciones descriptor-clasificador utilizando los modelos nombrados en la Tabla 3.2.

Para facilitar la lectura y posterior análisis comparativo de los datos, en la siguiente tabla se presentan las métricas individuales solo de las 6 combinaciones descriptor-clasificador con mejores resultados (la totalidad de los resultados individuales se pueden observar en las Tablas A.1, A.2, A.3, A.4).

	DeCAF7-SVC	ResNet152-SVC	DeCAF7-Random Forest	DeCAF7-Logistic Regresion	DeCAF7-Multilayer Perceptron	Histograma de Grises-Random Forest
Exactitud	**75 % ± 4 %**	72 % ± 5 %	73 % ± 3 %	72 % ± 5 %	73 % ± 4 %	71 % ± 5 %
Precisión	**75 % ± 4 %**	71 % ± 8 %	74 % ± 3 %	72 % ± 5 %	72 % ± 4 %	70 % ± 5 %
Recall	**75 % ± 3 %**	71 % ± 7 %	74 % ± 4 %	71 % ± 6 %	73 % ± 4 %	70 % ± 5 %
Puntuación F1	**74 % ± 5 %**	72 % ± 8 %	**74 % ± 2 %**	71 % ± 4 %	73 % ± 5 %	71 % ± 4 %

Tabla 4.1: Métricas de las mejores 6 combinaciones descriptor-clasificador

4.2. Experimento Undersampling

En esta sección se presentan los resultados obtenidos al probar los métodos de ensemble usando cada uno de los subconjuntos de modelos descriptor-clasificador descritos en el Capítulo 3.4.1, los cuales fueron entrenados con el dataset preprocesado por la técnica *undersampling* (ver Capítulo 3.3).

Subconjunto 1

	SVC	Random Forest	Gaussian Naive Bayes	Regresión Logística	Perceptrón Multicapa	K-Nearest Neighbors
Exactitud	66 % ± 2 %	**75 % ± 2 %**	54 % ± 14 %	66 % ± 4 %	61 % ± 7 %	67 % ± 3 %
Precisión	73 % ± 1 %	**75 % ± 1 %**	73 % ± 1 %	72 % ± 2 %	71 % ± 2 %	73 % ± 2 %
Recall	66 % ± 2 %	**75 % ± 2 %**	53 % ± 14 %	66 % ± 4 %	61 % ± 7 %	67 % ± 3 %
Puntuación F1	67 % ± 2 %	**75 % ± 2 %**	49 % ± 19 %	66 % ± 4 %	61 % ± 7 %	68 % ± 3 %

Tabla 4.2: Resultados método *Stacked Ensemble*, usando el subconjunto 1 de modelos

	SVC	Random Forest	Gaussian Naive Bayes	Logistic Regression	Multilayer Perceptron	K-Nearest Neighbors
Exactitud	**76 % ± 3 %**	**76 % ± 4 %**	68 % ± 2 %	**76 % ± 5 %**	75 % ± 3 %	75 % ± 4 %
Precisión	**76 % ± 2 %**	**76 % ± 4 %**	74 % ± 3 %	**76 % ± 5 %**	75 % ± 2 %	**76 % ± 4 %**
Recall	**76 % ± 3 %**	**76 % ± 4 %**	68 % ± 2 %	**76 % ± 4 %**	75 % ± 3 %	75 % ± 4 %
Puntuación F1	**76 % ± 5 %**	**76 % ± 4 %**	68 % ± 3 %	**76 % ± 4 %**	75 % ± 3 %	75 % ± 4 %

Tabla 4.3: Resultados método *One vs Rest Ensemble*, usando el subconjunto 1 de modelos

	Votación Mayoritaria	Votación Mayoritaria con Peso	Super Learner
Exactitud	73 % ± 2 %	74 % ± 3 %	74 % ± 2 %
Precisión	74 % ± 2 %	74 % ± 2 %	74 % ± 3 %
Recall	73 % ± 2 %	74 % ± 2 %	74 % ± 3 %
Puntuación F1	73 % ± 2 %	74 % ± 2 %	74 % ± 2 %

Tabla 4.4: Resultados ensembles de tipo *Voting*, usando subconjunto 1 de modelos

Subconjunto 2

	SVC	Random Forest	Gaussian Naive Bayes	Logistic Regression	Multilayer Perceptron	K-Nearest Neighbors
Exactitud	69 % ± 3 %	**76 % ± 3 %**	60 % ± 3 %	67 % ± 2 %	65 % ± 4 %	70 % ± 4 %
Precisión	73 % ± 3 %	**76 % ± 2 %**	74 % ± 1 %	71 % ± 2 %	69 % ± 2 %	73 % ± 3 %
Recall	69 % ± 4 %	**76 % ± 3 %**	59 % ± 9 %	67 % ± 2 %	65 % ± 4 %	69 % ± 4 %
Puntuación F1	70 % ± 4 %	**75 % ± 2 %**	58 % ± 10 %	67 % ± 2 %	65 % ± 4 %	70 % ± 4 %

Tabla 4.5: Resultados método *Stacked Ensemble*, usando el subconjunto 2 de modelos

	SVC	Random Forest	Gaussian Naive Bayes	Logistic Regression	Multilayer Perceptron	K-Nearest Neighbors
Exactitud	**76 % ± 3 %**	**76 % ± 3 %**	71 % ± 3 %	**76 % ± 2 %**	**76 % ± 3 %**	**76 % ± 2 %**
Precisión	**77 % ± 3 %**	76 % ± 3 %	73 % ± 3 %	76 % ± 3 %	76 % ± 2 %	76 % ± 2 %
Recall	**76 % ± 2 %**	**76 % ± 3 %**	71 % ± 3 %	**76 % ± 2 %**	**76 % ± 3 %**	**76 % ± 2 %**
Puntuación F1	**76 % ± 3 %**	**76 % ± 3 %**	71 % ± 4 %	**76 % ± 2 %**	**76 % ± 2 %**	**76 % ± 2 %**

Tabla 4.6: Resultados método *One vs Rest Ensemble*, usando el subconjunto 2 de modelos

	Votación Mayoritaria	Votación Mayoritaria con Peso	Super Learner
Exactitud	75 % ± 3 %	75 % ± 2 %	75 % ± 2 %
Precisión	75 % ± 2 %	76 % ± 2 %	75 % ± 2 %
Recall	75 % ± 3 %	75 % ± 2 %	75 % ± 2 %
Puntuación F1	75 % ± 3 %	75 % ± 2 %	75 % ± 2 %

Tabla 4.7: Resultados ensembles de tipo *Voting*, usando subconjunto 2 de modelos

Subconjunto 3

	SVC	Random Forest	Gaussian Naive Bayes	Logistic Regression	Multilayer Perceptron	K-Nearest Neighbors
Exactitud	68 % ± 3 %	**76 % ± 2 %**	58 % ± 7 %	68 % ± 2 %	67 % ± 4 %	71 % ± 4 %
Precisión	73 % ± 2 %	**76 % ± 1 %**	74 % ± 1 %	72 % ± 1 %	72 % ± 2 %	74 % ± 3 %
Recall	68 % ± 3 %	**76 % ± 2 %**	58 % ± 8 %	68 % ± 2 %	67 % ± 4 %	71 % ± 4 %
Puntuación F1	69 % ± 3 %	**75 % ± 2 %**	57 % ± 9 %	69 % ± 2 %	67 % ± 4 %	71 % ± 4 %

Tabla 4.8: Resultados método *Stacked Ensemble*, usando el subconjunto 3 de modelos

	SVC	Random Forest	Gaussian Naive Bayes	Logistic Regression	Multilayer Perceptron	K-Nearest Neighbors
Exactitud	**77 % ± 2 %**	71 % ± 4 %	71 % ± 4 %	**77 % ± 2 %**	76 % ± 2 %	76 % ± 2 %
Precisión	**77 % ± 2 %**	75 % ± 2 %	75 % ± 1 %	**77 % ± 3 %**	76 % ± 2 %	**77 % ± 3 %**
Recall	**77 % ± 2 %**	71 % ± 4 %	71 % ± 4 %	**77 % ± 2 %**	76 % ± 2 %	76 % ± 3 %
Puntuación F1	**77 % ± 2 %**	71 % ± 5 %	71 % ± 4 %	**77 % ± 2 %**	76 % ± 2 %	76 % ± 2 %

Tabla 4.9: Resultados método *One vs Rest Ensemble*, usando el subconjunto 3 de modelos

	Votación Mayoritaria	Votación Mayoritaria con Peso	Super Learner
Exactitud	75 % ± 2 %	76 % ± 2 %	76 % ± 2 %
Precisión	75 % ± 3 %	76 % ± 2 %	76 % ± 2 %
Recall	75 % ± 2 %	76 % ± 2 %	76 % ± 2 %
Puntuación F1	75 % ± 2 %	76 % ± 2 %	76 % ± 2 %

Tabla 4.10: Resultados ensembles de tipo *Voting*, usando subconjunto 3 de modelos

4.2.1. Resultados

De acuerdo a los resultados presentados para este primer experimento, se puede observar como, en su gran mayoría, las distintas métricas se mantienen cercanas al 75 %, alcanzando un máximo en el 77 %, el cual se obtiene con el método *One vs Rest Ensemble* en el subconjunto 3 y usando Clasificador de Vectores de Soporte (SVC) y Regresión Logística (*Logistic Regression*) como meta-clasificadores. Además de estos modelos mencionados, también destacan las métricas obtenidas con *One vs Rest* en el subconjunto 2 al usar SVC como meta-clasificador y en el subconjunto 3 al usar *K-Nearest Neighbors*, en ambos caso se alcanza una precisión del 77 %.

Por otro lado, los resultados obtenidos con el método *Stacked Ensemble* son, evaluados de forma general, los mas bajos, alcanzando el peor resultado al usar, como meta-clasificador, *Gaussian Naive Bayes* en el subconjunto 1 y obteniendo una exactitud del 54 % y una puntuación F1 del 49 %. Aunque este método de ensemble presenta alguno de los resultados más bajos, también se obtienen resultados equiparables con los ya mencionados como más altos, esto ocurre, por ejemplo, al usar *Random Forest* y el subconjunto 2, logrando alcanzar un 76 % en todas las métricas.

Finalmente, los métodos de ensemble de tipo *Voting* se mantienen por sobre un 70 % en sus métricas, siendo el método de Votación Mayoritaria él con resultados más bajos, alcanzando el peor en el subconjunto 1, donde alcanza una exactitud del 73 % y una precisión del 74 %. Por otro lado, los métodos Votación Mayoritaria con Peso y *Super Learner* obtienen resultados muy similares y ambos logran alcanzar un 76 % en todas sus métricas al usar el subconjunto 3.

4.2.1.1. Análisis

Comparando esto resultados con los mostrados en la sección 4.1, se obtiene una leve mejora en las métricas, siendo un 75 % la mejor precisión de los resultados previos y, obteniéndose, un 77 % al usar el ensemble *One vs Rest* con *Logistic Regression* como meta clasificador. Además de esta mejora en las métricas, otra característica que se obtiene al usar los métodos de ensemble, es la disminución de la varianza, en los casos mencionados donde se dan los mejores resultados, la varianza se mantiene en el 2 %, alcanzando un 3 % como máximo. La baja varianza permite inferir que los modelos entregan resultados similares ante distintos conjuntos de entrenamiento y de prueba, lo cual genera una mayor confianza en el modelo al probar su funcionamiento en nuevos datos de entrada.

Por otro lado, es importante notar que al comparar los dos métodos de ensemble de tipo *stacking*, *Stacked Ensemble* y *One vs Rest Ensemble*, el segundo, aparte de obtener mejores resultados en su mejor caso, también mantiene las métricas obtenidas en valores altos con todos los meta-clasificadores que se usaron, algo que no ocurre con *Stacked Ensemble*, y que puede indicar que el método *One vs Rest Ensemble* se adapta de mejor forma, al menos a este caso de estudio.

4.3. Experimento Cost-sensitive

En esta sección se presentan los resultados obtenidos, al entrenar los modelos descriptor-clasificador con la técnica de entrenamiento *Cost-sensitive* (ver Capítulo 3.4.2). Los resultados se presentan separados en tres divisiones, la primera muestra los resultados de un clasificador individual que solo elige como predicción la clase con mayor cantidad de ejemplos (No-Meme), la segunda presenta las métricas obtenidas de forma individual por cada una de las combinaciones descriptor-clasificador y por último, en la tercera división se muestran los resultados de los modelos basados en ensembles.

Clasificador Clase Mayoritaria

	Exactitud	Precisión	Recall	Puntuación F1
Clasificador	95 % ± 0 %	32 % ± 0 %	33 % ± 0 %	32 % ± 0 %

Tabla 4.11: Métricas modelo que siempre entrega como predicción la clase mayoritaria

Resultados Individuales

	Histograma de Grises	Distribución de Color	Histograma de Gradientes Orientados	Histograma de Bordes	ResNet152	DeCAF7
Exactitud	70 % ± 4 %	41 % ± 3 %	59 % ± 2 %	52 % ± 2 %	70 % ± 2 %	65 % ± 1 %
Precisión	39 % ± 1 %	35 % ± 0.1 %	37 % ± 0.3 %	36 % ± 0.2 %	39 % ± 1 %	41 % ± 0.4
Recall	61 % ± 2 %	49 % ± 2	59 % ± 2 %	54 % ± 1 %	44 % ± 1 %	51 % ± 1 %
Puntuación F1	38 % ± 2 %	24 % ± 1 %	33 % ± 1 %	29 % ± 1 %	37 % ± 1 %	35 % ± 1 %

Tabla 4.12: Resultados modelos individuales *cost-sensitive*, usando SVC como clasificador

	Histograma de Grises	Distribución de Color	Histograma de Gradientes Orientados	Histograma de Bordes	ResNet152	DeCAF7
Exactitud	95 % ± 0.5 %	95 % ± 0.1 %	95 % ± 0.1 %	95 % ± 0.1 %	95 % ± 0.1 %	95 % ± 0.1 %
Precisión	71 % ± 7 %	66 % ± 2 %	57 % ± 14 %	56 % ± 1 %	51 % ± 14 %	75 % ± 2 %
Recall	37 % ± 1 %	36 % ± 1 %	34 % ± 0.3 %	35 % ± 0.4 %	34 % ± 0.4 %	37 % ± 1 %
Puntuación F1	39 % ± 2 %	37 % ± 2 %	35 % ± 0.6 %	35 % ± 1 %	34 % ± 0.5 %	40 % ± 1 %

Tabla 4.13: Resultados modelos individuales *cost-sensitive*, usando *Random Forest* como clasificador

	Histograma de Grises	Distribución de Color	Histograma de Gradientes Orientados	Histograma de Bordes	ResNet152	DeCAF7
Exactitud	84 % ± 1 %	45 % ± 4 %	57 % ± 3 %	54 % ± 2 %	90 % ± 0.3 %	90 % ± 0.3 %
Precisión	41 % ± 1 %	35 % ± 0.3 %	37 % ± 0.3 %	36 % ± 3 %	48 % ± 1 %	50 % ± 1 %
Recall	55 % ± 2 %	47 % ± 2 %	57 % ± 2 %	53 % ± 2 %	62 % ± 2 %	70 % ± 2 %
Puntuación F1	43 % ± 1 %	26 % ± 1 %	32 % ± 1 %	30 % ± 1 %	52 % ± 1 %	56 % ± 1 %

Tabla 4.14: Resultados modelos individuales *cost-sensitive*, usando *Logistic Regression* como clasificador

Resultados Ensembles

	SVC	Random Forest	Logistic Regression
Exactitud	95 % ± 0.1 %	95 % ± 0.1 %	95 % ± 0.1 %
Precisión	66 % ± 9 %	62 % ± 6 %	58 % ± 9 %
Recall	36 % ± 0.5 %	35 % ± 0.4 %	35 % ± 0.4 %
Puntuación F1	38 % ± 1 %	36 % ± 0.8 %	35 % ± 1 %

Tabla 4.15: Resultados método *Stacked Ensemble* en Experimento Cost-sensitive

	SVC	Random Forest	Logistic Regression
Exactitud	95 % ± 0.1 %	95 % ± 0.1 %	95 % ± 0.1 %
Precisión	67 % ± 4 %	63 % ± 5 %	62 % ± 6 %
Recall	36 % ± 0.5 %	35 % ± 0.3 %	35 % ± 0.2 %
Puntuación F1	38 % ± 1 %	36 % ± 0.8 %	36 % ± 0.3 %

Tabla 4.16: Resultados método *One vs Rest Ensemble* en Experimento Cost-sensitive

	Votación Mayoritaria	Votación Mayoritaria con Peso	Super Learner
Exactitud	95 % ± 0.1 %	96 % ± 0.2 %	96 % ± 0.2 %
Precisión	66 % ± 4 %	72 % ± 5 %	72 % ± 5 %
Recall	54 % ± 0.7 %	50 % ± 1 %	49 % ± 1 %
Puntuación F1	59 % ± 2 %	56 % ± 1 %	56 % ± 1 %

Tabla 4.17: Resultados ensembles de tipo *Voting* en Experimento Cost-sensitive

Resumen Mejores Resultados

	Individual: DeCAF7-Random Forest	**Stacking:** One vs Rest - SVC	**Voting:** Votación Mayoritaria con Peso
Exactitud	95 % ± 0.1 %	95 % ± 0.1 %	96 % ± 0.2 %
Precisión	75 % ± 2 %	67 % ± 4 %	72 % ± 5 %
Recall	37 % ± 1 %	36 % ± 0.5 %	50 % ± 1 %
Puntuación F1	40 % ± 1 %	38 % ± 1 %	56 % ± 1 %

Tabla 4.18: Mejores Resultados Experimento *Cost-Sensitive*

4.3.1. Resultados

Los resultados individuales muestran como el clasificador *Random Forest* presenta los mejores resultados generales, al combinarlo con el descriptor DeCAF7 obtiene los mayores valores, tanto en exactitud como en precisión, siendo estos 95 % y 75 % respectivamente, pero que cae al 37 % en *recall*. Por otro lado, el modelo formado por DeCAF7 y *Logistic Regression* es el que presenta mayor puntuación F1 y *recall*, manteniendo una exactitud del 90 %.

En el caso de los modelos de ensemble de tipo *Stacking*, los resultados entre el método *Stacked* y *One vs Rest*, al usar un mismo meta clasificador, se muestran sin mucha variación entre sí, obteniéndose las mejores métricas al usar *One vs Rest* con SVC, llegando a alcanzar el 95 % de exactitud, 67 % de precisión, 36 % de *recall* y 38 % de puntuación F1. Para los ensembles de tipo *voting*, los mejores resultados se alcanzan al usar *Super Learner* y Votación Mayoritaria con Peso, siendo este último mínimamente mejor al obtener un 50 % de *recall* por sobre el 49 % de *Super Learner*, además, en ambos casos, es importante destacar que logran alcanzar una exactitud del 96 %.

4.3.1.1. Análisis

Resultados a destacar, y que servirán como cota inferior para analizar el desempeño del resto de modelos, son los obtenidos por el clasificador que solo entrega como predicción la clase mayoritaria (No Meme), el cual obtiene una exactitud del 95 %, lo cual concuerda con el hecho de que el 95 % de las imágenes en el dataset corresponden a la clase No Meme. Por otro lado, las demás métricas se ubican en el 32 % para la precisión y la puntuación F1 y

un 33 % para el *recall*. Nuevamente, estos resultados concuerdan con lo esperado, debido a que el cálculo de estas métricas es la media aritmética de lo obtenido para cada una de las clases, esto implica que si para la clase No Meme una métrica es un 100 % y para el resto de clases es un 0 %, el resultado final sería un 33.33 %, lo cual es justo lo que ocurre en el caso del *recall*. En el caso de las métricas precisión y puntuación F1, se obtiene un 32 % debido a que la precisión penaliza los falsos positivos, lo que lleva a que su valor en la clase No Meme no llegue al 100 % exacto, este hecho también afecta a la puntuación F1 dado que su cálculo depende de la precisión.

Dado el gran desbalance en las clases y que el clasificador Clase Mayor obtiene una exactitud del 95 % solo entregando como predicción la clase mayoritaria, un rendimiento esperable para el resto de modelos entrenados con la técnica *cost-sensitive*, era que sus métricas fueran similares a las mostradas en la Tabla 4.11, implicando que los clasificadores al entrenarse y optimizar resultados se convirtieran en una especie de Clasificador Clase Mayoritaria, siendo esta la forma más simple de maximizar la exactitud. Pero, observando lo obtenido, esto no ocurre, en general, los resultados son muy variados y distintos a los presentados en la Tabla 4.11.

Con respecto al mejor resultado de los modelos individuales, obtenido por la combinación de DeCAF7 y *Random Forest*, aunque obtiene excelentes valores para la exactitud y precisión, su *recall* no supera el 37 %, esto se debe principalmente a que esta métrica penaliza los falsos negativos, sobre todo al producirse en clases con pocos ejemplos. De forma similar ocurre en casos en donde el *recall* obtiene buenos resultados, pero la precisión disminuye bastante, esto se puede deber a que los clasificadores al intentar cometer menos falsos negativos en las clases con menos ejemplos, incrementan los resultados que dan como predicción a estas clases, lo que termina provocando un aumento en los falsos positivos y, con ello, una disminución en la precisión.

Con todo esto, es posible deducir que el mejor modelo para este experimento, en términos generales, es el ensemble de Votación Mayoritaria con Peso, el cual logra la máxima exactitud y obtiene buenos valores en la precisión, sin sacrificar demasiado el *recall*. Esto se observa más claro al ver que su puntuación F1 (la cual es una ponderación armónica de la precisión y el *recall*) es la segunda más alta, solo superada por el ensemble que usa Votación Mayoritaria.

4.4. Modificaciones One vs Rest Ensemble

Debido a las mejoras que se lograron al usar el ensemble *One vs Rest*, se intentó experimentar con este método buscando mejorar los resultados, para ello se realizaron dos experimentos, el primero buscaba modificar el entrenamiento de los modelos y el segundo modificaba los modelos usados en el ensemble.

Entrenamiento Modificado

En esta versión de *One vs Rest Ensemble* se modificó el entrenamiento. Como se puede observar en el Capítulo 2.6.4, los modelos se usan tres veces para lograr una aproximación *one vs rest*, usando una función indicadora para ello. En esta versión modificada, se entrenó cada modelo, una vez por cada clase posible, para lograr la clasificación binaria (1 si pertenece a

la clase, 0 en otro caso) sin usar la función indicadora.

Los resultados obtenidos con este método no consiguieron una mejora en lo obtenido de forma normal. Las métricas resultantes rondaban entre el 74 % y 75 % de precisión, lo cual, aunque se puede considerar como un resultado alto, no supera los 77 alcanzados en el método original usando el subconjunto 3 de modelos.

Conjuntos de Modelos Modificados

Como se observa en la Figura 2.12, existen tres conjuntos de modelos que generan los meta datos. De forma normal, cada uno de estos conjuntos contiene los mismos modelos, y su predicción se diferencia a través de la función indicadora, en esta versión de *One vs Rest Ensemble* se cambió este hecho, generando tres conjuntos distintos, los cuales contenían, cada uno, los diez mejores modelos tomando en cuenta únicamente su resultado para la clase asignada al conjunto. Los resultados analizados para la obtención de los diez mejores modelos por clase fueron las matrices de confusión disponibles en el estado del arte, específicamente en el trabajo de memoria "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020 [1].

Con esta modificación, los resultados obtenidos alcanzan, pero sin superar, lo logrado usando *One vs Rest Ensemble* de forma normal, es decir, se alcanza, en cada una de las métricas, un valor del 77 %, lo cual sigue siendo un buen resultado, pero para lo cual se necesita que se escoja de forma manual cada uno de los tres conjuntos de modelos que usa este método de ensemble.

Capítulo 5

Conclusiones

El trabajo realizado en esta memoria permite explorar el funcionamiento de distintos métodos de ensemble en el problema de clasificación de imágenes en las clases No Meme, Meme o Sticker. Además, se amplía el estudio del problema de clasificación presentado, añadiendo nuevos resultados tanto al usar ensembles con los modelos disponibles en el estado del arte, como al reentrenar estos modelos con la técnica *Cost-sensitive*, lo cual permite aprovechar de mejor forma el dataset del cual se dispone.

Los resultados que se obtuvieron en el experimento *undersampling* y que son comparables con los presentes en el estado del arte, muestran una pequeña mejora en las métricas resultantes, pasando de un 75 % de precisión en los valores máximos de los modelos individuales a un 77 % con el uso de ensembles. Este resultado, aunque pueda verse como positivo, también plantea ciertos problemas, pese a que pueda parecer que usar el ensemble es una mejor opción, el costo de entrenar y probar este método es varias veces mayor que el del modelo descriptor-clasificador en solitario, esto en datasets no muy grandes puede despreciarse, pero al usar un conjunto de entrenamiento con una gran cantidad de ejemplos el tiempo que demore el entrenamiento puede ser un punto importante para decidir si usar el ensemble o no.

Por otra parte, los resultados percibidos en el experimento *cost-sensitive*, se observan como un avance positivo, permitiendo añadir nuevos resultados al estudio de este problema de clasificacion. El que no existan resultados previos en el estado del arte vuelve complicado determinar si las métricas obtenidas en este trabajo, para este experimento pueden considerarse buenas o no. En general, aunque el mejor modelo logra alcanzar altos valores de exactitud y precisión, el *recall* no logra buenos resultados, esto, aunque malo para la meta de este trabajo, es bueno para el estado del arte, dado que genera un punto que mejorar en el que futuros trabajos sobre el tema puedan indagar.

5.1. Limitaciones y trabajo futuro

El experimento *undersampling* logra sus mejores resultados al usar el subconjunto 2 de modelos, este subconjunto contiene las 18 mejores combinaciones descriptor-clasificador, a las cuales se les agregan 4 modelos extras. El hecho de agregar los 4 modelos extras, busca aumentar variabilidad a los resultados individuales y que se traduzcan en mejoras para los ensembles, esto parece funcionar en cierta forma, obteniéndose mínimas mejoras, pero este

resultado también plantea la duda de si existe algún subconjunto de modelos especifico donde, para cierta configuración de ensemble, se maximicen los resultados. Buscar este subconjunto no es trivial dado que se tienen 36 combinaciones descriptor-clasificador, lo que genera un total de $2^{36} = 68.719.476.736$ subconjuntos posibles, debido a esto buscar el mejor subconjunto pareciera ser una tarea extremadamente difícil, pero se plantea su búsqueda como un trabajo futuro que busque mejorar los resultados obtenidos en esta memoria.

Por otro lado, y como se menciona en el trabajo de memoria "Comparación de descriptores para clasificación de memes" [1], los descriptores usados como base para la generación de vectores de características que los diferentes clasificadores puedan leer, solo extraen la información visual de la imagen, otro contenido, como por ejemplo el textual, no es analizado por estos descriptores, esto puede ser determinante al momento de clasificar una imagen, sobre todo al diferenciar entre meme y sticker, donde el contenido textual juega un rol de suma importancia para la correcta predicción. Con esto, se plantea como trabajo futuro el uso de algún descriptor que permita extraer el texto de una imagen, combinado con técnicas de Procesamiento del Lenguaje Natural (NLP) [25], para generar un vector de características y probar tanto el desempeño de los clasificadores como el de los ensembles.

Bibliografía

[1] A. Sara Zaror, "Comparación de descriptores para clasificación de memes," Memoria de Ingeniería Civil en Computación, Universidad de Chile, 2020.

[2] S. Laencina Verdaguer, *Color based image classification and description*. PhD thesis, UPC, Escola Politècnica Superior de Castelldefels, Oct 2009.

[3] B. Manjunath, J. Ohm, V. Vasudevan, and A. Yamada, "Color and texture descriptors," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 11, pp. 703 – 715, 07 2001.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, 2005.

[5] M. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—A review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, p. I–647–I–655, JMLR.org, 2014.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 770–778, IEEE Computer Society, jun 2016.

[8] T. Evgeniou and M. Pontil, *Support Vector Machines: Theory and Applications*, pp. 249–257. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, p. 5–32, Oct. 2001.

[10] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, *k-Nearest Neighbor Classification*, pp. 83–106. New York, NY: Springer New York, 2009.

[11] C. Sammut and G. I. Webb, eds., *Logistic Regression*, pp. 631–631. Boston, MA: Springer US, 2010.

[12] H. Zhang, "The optimality of naive bayes," in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA* (V. Barr and Z. Markov, eds.), pp. 562–567, AAAI Press, 2004.

[13] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, pp. 1–39, Feb 2010.

[14] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241 – 259, 1992.

[15] J. Vanschoren, *Meta-learning*, pp. 39–61. Germany: Springer, 2019.

[16] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, Aug 1996.

[17] P. Bühlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science*, vol. 22, 05 2008.

[18] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, (Berlin, Heidelberg), pp. 1–15, Springer Berlin Heidelberg, 2000.

[19] S. Kang, S. Cho, and P. Kang, "Multi-class classification via heterogeneous ensemble of one-class classifiers," *Engineering Applications of Artificial Intelligence*, vol. 43, pp. 35 – 43, 2015.

[20] E. C. Polley and M. J. Van der Laan, "Super learner in prediction," 2010.

[21] C. O'Connor and H. Joffe, "Intercoder reliability in qualitative research: Debates and practical guidelines," *International Journal of Qualitative Methods*, vol. 19, p. 1609406919899220, 2020.

[22] "Encyclopedia of survey research methods au - lavrakas, paul," Apr 2008.

[23] D. M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, pp. 1–12, Jan 2004.

[24] 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*, vol. 12, pp. 2825–2830, 2011.

[25] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, 2003.

Anexo A

Tablas

	SVM	Random Forest	Naive Bayes Gausiano	Regresión Logística	Perceptrón Multicapa	K-Nearest Neighbors
Gris	64 % ± 6 %	70 % ± 5 %	51 % ± 5 %	53 % ± 8 %	59 % ± 5 %	54 % ± 6 %
Color	53 % ± 5 %	56 % ± 5 %	44 % ± 7 %	46 % ± 5 %	52 % ± 4 %	47 % ± 5 %
HoG	61 % ± 8 %	61 % ± 9 %	53 % ± 7 %	58 % ± 9 %	58 % ± 6 %	53 % ± 7 %
Bordes	57 % ± 7 %	56 % ± 5 %	49 % ± 5 %	53 % ±6 %	55 % ± 6 %	53 % ± 5 %
ResNet152	71 % ± 8 %	63 % ± 5 %	64 % ± 7 %	62 % ± 8 %	69 % ± 7 %	60 % ± 7 %
DeCAF7	75 % ± 4 %	74 % ± 3 %	65 % ± 4 %	72 % ± 4 %	72 % ± 4 %	65 % ± 5 %

Tabla A.1: Tabla de precisiones extraída de "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020, p. 32 [1]

	SVM	Random Forest	Naive Bayes Gausiano	Regresión Logística	Perceptrón Multicapa	K-Nearest Neighbors
Gris	64 % ± 7 %	71 % ± 5 %	52 % ± 5 %	52 % ± 8 %	59 % ± 6 %	53 % ± 5 %
Color	58 % ± 7 %	57 % ± 7 %	45 % ± 8 %	44 % ± 5 %	51 % ± 5 %	46 % ± 6 %
HoG	62 % ± 9 %	60 % ± 11 %	53 % ± 8 %	58 % ± 8 %	59 % ± 6 %	52 % ± 7 %
Bordes	56 % ± 7 %	57 % ± 5 %	50 % ± 5 %	51 % ±6 %	54 % ± 6 %	52 % ± 5 %
ResNet152	72 % ± 5 %	63 % ± 3 %	65 % ± 7 %	63 % ± 8 %	68 % ± 7 %	62 % ± 5 %
DeCAF7	75 % ± 4 %	73 % ± 3 %	66 % ± 5 %	72 % ± 5 %	73 % ± 4 %	65 % ± 5 %

Tabla A.2: Tabla de exactitud extraída de "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020, p. 32 [1]

	SVM	Random Forest	Naive Bayes Gausiano	Regresión Logística	Perceptrón Multicapa	K-Nearest Neighbors
Gris	63 % ± 7 %	70 % ± 5 %	50 % ± 5 %	53 % ± 7 %	59 % ± 6 %	55 % ± 6 %
Color	52 % ± 4 %	56 % ± 6 %	45 % ± 8 %	46 % ± 6 %	52 % ± 5 %	47 % ± 6 %
HoG	62 % ± 8 %	62 % ± 8 %	53 % ± 5 %	59 % ± 8 %	58 % ± 5 %	54 % ± 7 %
Bordes	56 % ± 7 %	57 % ± 6 %	49 % ± 6 %	51 % ±7 %	55 % ± 8 %	54 % ± 6 %
ResNet152	71 % ± 7 %	63 % ± 6 %	63 % ± 6 %	64 % ± 7 %	70 % ± 7 %	61 % ± 7 %
DeCAF7	75 % ± 3 %	74 % ± 4 %	65 % ± 5 %	71 % ± 6 %	73 % ± 4 %	65 % ± 6 %

Tabla A.3: Tabla de recall extraída de "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020, p. 32 [1]

	SVM	Random Forest	Naive Bayes Gausiano	Regresión Logística	Perceptrón Multicapa	K-Nearest Neighbors
Gris	65 % ± 5 %	71 % ± 4 %	50 % ± 5 %	53 % ± 8 %	60 % ± 5 %	55 % ± 5 %
Color	54 % ± 5 %	55 % ± 5 %	45 % ± 7 %	45 % ± 5 %	53 % ± 3 %	48 % ± 5 %
HoG	60 % ± 7 %	62 % ± 8 %	54 % ± 7 %	59 % ± 8 %	58 % ± 7 %	54 % ± 8 %
Bordes	56 % ± 7 %	57 % ± 5 %	50 % ± 5 %	52 % ± 5 %	55 % ± 5 %	54 % ± 5 %
ResNet152	72 % ± 8 %	64 % ± 5 %	65 % ± 6 %	61 % ± 6 %	70 % ± 6 %	62 % ± 6 %
DeCAF7	74 % ± 5 %	74 % ± 2 %	66 % ± 4 %	71 % ± 4 %	73 % ± 5 %	66 % ± 4 %

Tabla A.4: Tabla de F1 extraída de "Comparación de descriptores para clasificación de memes" de A. Sara Zaror 2020, p. 33 [1]