Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC6205 Procesamiento de Lenguaje Natural

Clasificación de Intensidad de Emociones

Competencia #1

Estudiantes:
Mario Vicuña
Miguel Videla

Equipo:
TeamChalla

Profesor:
Felipe Bravo

Auxiliares:
Pablo Badilla
Gabriel Chaperón
Cristián Tamblay

Fecha:
24 de Mayo de 2020

Índice

1. Introducción

La clasificación o categorización de intensidad de sentimientos en texto representa uno de los problemas más comunes en el área del procesamiento de lenguaje natural, el cual permite desarrollar análisis de sentimientos para distintos fines, tales como, estimación de resultados electorales, medición de grado de recepción o satisfacción de un conjunto objetivo de individuos frente a una determinada política pública o producto, medición de impactos de algún suceso en la población, apoyo en motores de búsqueda, etc.

El presente problema consiste en la clasificación de 3 niveles de intensidad de emociones (*low*, *medium*, *high*) de tweets divididos en 4 categorías de emoción diferentes (*anger*, *fear*, *joy*, *sadness*), en el contexto de una competencia de clasificación, donde se provee de un conjunto de entrenamiento para diseño y evaluación de distintas soluciones con el objetivo de obtener el mejor desempeño de clasificación de acuerdo a las métricas establecidas, sobre un conjunto de prueba sin etiquetas.

Se exploró la utilización de distintos algoritmos de machine learning para la clasificación de tweets, así como también, distintos métodos de representación de características de cada uno, analizando el desempeño de cada configuración en base a 3 métricas de desempeño (*accuracy*, *kappa* y AUC). Entre las representaciones con las que se experimento se encuentran los n-gramas, diversos contadores, utilización de *lexicons* y *embeddings*, mientras que entre los algoritmos de machine learning utilizados se encuentran *naïve Bayes*, *support vector machines*, *random forest*, *k-nearest neighbors*, perceptrones multicapa, redes neuronales recurrentes y *transformers*, explorando adicionalmente ensamblaje de clasificadores mediante la técnica de *soft voting*.

El mejor desempeño fue obtenido mediante la utilización de un *embedding* mediante el modelo de *deep learning* del estado del arte en procesamiento de lenguaje natural basado en *transformers* BERT junto con un ensamblaje de 10 clasificadores *random forest* mediante *soft voting*, obteniendo el primer lugar en promedio en cada métrica de la competencia, con 0.79 en AUC, 0.442 en coeficiente *kappa* y 0.684 en *accuracy*.

Se evidencia la alta dificultad de la tarea de clasificación en el contexto de lenguaje natural, constatando la relevancia del conocimiento experto y de un estudio profundo del conjunto de datos utilizado para la selección de características relevantes y algoritmos de clasificación adecuados bajo el enfoque de *machine learning* clásico, además de comprobar la efectividad de algoritmos modernos basados en *deep learning* para suplir la etapa de ingeniería de características mediante extracción de patrones inherentes en los datos, trasladando el proceso de diseño la construcción de una arquitectura para el modelamiento adecuado del problema, la cual, dependiendo de la calidad y cantidad de datos disponibles, es capaz de superar el desempeño obtenido mediante diseño manual de características, cuando se carece de la expertíz necesaria en el área.

2. Métricas de Evaluación

En el presente proyecto se consideran varias métricas de evaluación, las cuales en el contexto de la competencia pueden ser divididas en dos categorías según su uso.

Por un lado se encuentran las métricas que no son directamente consideradas, en el sentido que en la competencia no serán usadas como un criterio de evaluación. No obstante, en el proceso de diseño y experimentación estas pueden ser consideradas para mejorar los algoritmos estudiados. Dentro de esta categoría se encuentran:

- **Precision**: Corresponde a la fracción de clasificaciones correctas entre los datos clasificados como positivos. Matemáticamente Precision corresponde a:

$$P = \frac{TP}{TP + FP} \tag{1}$$

  siendo TP la cantidad de clasificaciones positivas correctas y FP incorrectas. Conceptualmente, optimizar sobre la Precision corresponde a minimizar el Error de Tipo I, es decir que se busca una baja tasa de falsas detecciones.

- **Recall**: Corresponde a la fracción de clasificaciones correctas entre los datos que verdaderamente corresponden a la clase positiva. Matemáticamente Recall corresponde a:

$$R = \frac{TP}{TP + FN} \tag{2}$$

  siendo FN los falsos negativos o datos de clase positiva incorrectamente clasificados. Conceptualmente, optimizar sobre el Recall corresponde a minimizar el Error de Tipo II, es decir que se busca una baja tasa de no-detecciones.

Típicamente existe un trade-off entre las métricas de Precision y Recall. Por ejemplo, para maximizar el Recall un algoritmo podría determinar que todas sus entradas son postivas (con lo que $FN = 0$). Evidentemente con esto se dispara la tasa de falsos positivos, perjudicando evidentemente la Precision del modelo y, más aún, es un comportamiento que claramente no se desea en un clasificador.

No obstante, se pueden combinar ambas métricas, por ejemplo mediante el F1-Score, la cual es la tercera medida de desempeño en el grupo de las que no se evaluarán en la competencia.

- **F1-Score**: El F1-Score surge como una métrica que combina ambas con la finalidad de aproximarse a un punto óptimo, por medio de una media harmónica ponderada mediante un parámetro de balance α. Esta métrica, en términos de P y R se define matemáticamente como:

$$F_\alpha = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha)\frac{1}{R}} \tag{3}$$

También se puede escribir esta expresión en términos del factor de balance β de la siguiente forma:

$$F_\beta = (1 + \beta^2)\frac{PR}{\beta^2 P + R} \tag{4}$$

Cuando $\beta = 1$, o equivalentemente $\alpha = \frac{1}{2}$, $F_\beta = F_1$ toma precisamente la forma de la media harmónica (balanceada) entre P y R, de ahí que esta métrica se conozca como F1-Score. En términos generales se la denomina simplemente F-Score, pudiendo considerar cualquier valor para α o β.

Por otra parte, entre las métricas que efectivamente serán consideradas para medir el desempeño de cada participante en la competencia son:

- ***Accuracy***: Esta es otra métrica "clásica" para algoritmos de clasificación, y corresponde a la razón de clasificaciones correctas (tanto positivas como negativas). En este sentido, se busca que el algoritmo acierte tanto al discriminar tanto los casos positivos como los negativos, por lo que la métrica a maximizar es:

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

  Planteado de esta forma pareciera que es evidente que esta métrica cumple con lo que intuitivamente se esperaría de un sistema de clasificación/detección, al minimizar las equivocaciones **en general**, tanto para los casos de clase positiva como negativa. No obstante para datos desbalanceados podría incurrirse en errores como el dado de ejemplo para el caso del Recall. Por ejemplo, si los casos de clase positiva son escasos, un algoritmo de clasificación puede lograr altos valores de *Accuracy* catalogando todos los casos como negativos (la que sería la clase mayoritaria). Este tipo de análisis y consideraciones toma especial relevancia al momento de tratar problemas de varias clases o de la vida real, como diagnósticos médicos.

- **AUC**: Para entender la métrica de AUC (Area under Curve) es necesario introducir primero el concepto de Curva ROC. A grandes rasgos, la curva ROC captura el rendiento del modelo en distintos regímenes o umbrales entre la tasa de falsos positivos y la tasa de verdaderos positivos, como se ve en la siguiente figura ilustrativa:

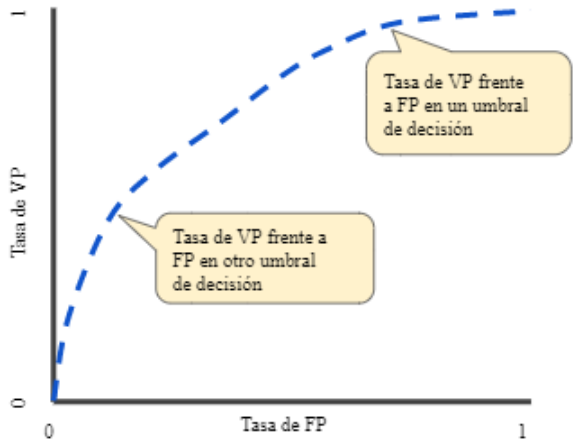

Figura 1: Figura ilustrativa de una curva ROC.

Un régimen se define a partir del umbral por el cual se toma una decisión, por ejemplo si la probabilidad de pertenecer a la clase positiva es mayor a tal umbral, se califica como perteneciente a dicha clase y como negativa en caso contrario. Por lo tanto, al cambiar dicho umbral se alteran las capacidades de distinguir una clase de otra. No obstante, la forma de la curva no depende solo del umbral de decisión; de hecho para un umbral dado solo se tiene un punto de la curva y al cambiar dicho umbral dicho punto se traslada **a través** de la curva.

La forma de la curva ROC está dada por el test de hipótesis que se desea modelar. Para ilustrar esto sean dos distribuciones de probabilidad sobre el espacio de observaciones. Al muestrear un valor en dicho espacio, el propósito del problema de detección es determinar de qué distribución proviene dicha muestra. Naturalmente, cuando las distribuciones se solapan de manera leve el problema es sencillo puesto que en términos prácticos poseen soportes disjuntos. Por otro lado, si las distribuciones correspondientes a la clase positiva y negativa coinciden, el problema se hace indistinguible, pues una muestra dada es igualmente verosimil en ambos casos. Considerando estos dos ejemplos es posible inferir la forma que tiene la curva ROC según dicho traslape e interpretar a partir de los ejes:

- Cuando el problema es indistinguible (las distribuciones coinciden) trazar un umbral de decisión hace que se acepten como positivas tantas muestras positivas como negativas, por lo que existe una relación uno a uno entre la tasa de verdaderos positivos y la tasa de falsos positivos. En consecuencia, la curva ROC es diagonal de pendiente unitaria y los puntos sobre esta son de la forma $(x, x), x \in [0, 1]$.
- Cuando el problema es perfectamente separable, es decir que las distribuciones de ambas clases presentan muy poco traslape, un umbral de decisión permite rechazar la gran mayoría de muestras pertenecientes a la clase negativa, sin perjudicar la correcta detección de las de clase positiva. En tal caso, la curva ROC aproxima un escalón unitario con dominio en $[0, 1]$.

Acá se presenta una GUI **online** que muestra la curva ROC correspondiente dadas dos distribuciones Gaussianas para las clases y un umbral, donde se aprecian los comportamientos descritos.

En consecuencia, los mejores desempeños son obtenibles en los casos en que la curva ROC se aproxima a la función escalón. De aquí el interés de la métrica AUC: la curva ROC está "acotada" por el escalón cuya área es 1, por lo tanto a mayor área bajo la curva ROC, más similar es esta a su cota superior y es posible hacer detecciones más precisas.

Planteado de esta forma, esta es una métrica que depende solamente del problema estudiado, mientras que la dependencia con el sistema de decisión es muy poco sustancial. No obstante, muchos problemas en la práctica no cuentan con un modelo estadístico como el planteado, es decir que no se conoce ninguna de las dos distribuciones. Acá es justamente donde entran en juego los modelos de aprendizaje, pues estos .ªprenden las distribuciones.ª partir de los datos y en consecuencia es el propio algoritmo el que define la curva ROC al reconocer las distribuciones. Luego, el criterio para elegir un modelo basado en esta métrica es aquel que tras ser entrenado obtenga la mayor área bajo la curva (la cual se obtiene en forma numérica).

- **Kappa**: En términos generales, el Coeficiente Kappa (κ) de Cohen mide estadísticamente el grado de acuerdo entre dos examinadores/clasificadores, considerando el efecto del azar. En este sentido, se penaliza la probabilidad de que ambos clasificadores hayan determinado el mismo resultado por mero efecto estadístico más que por algún esquema de decisión subyacente. Para ilustrar esto analicemos la definición de esta métrica:

$$\kappa = \frac{\mathbb{P}(a) - \mathbb{P}(r)}{1 - \mathbb{P}(r)} \tag{6}$$

donde el evento a corresponde al acuerdo entre los examinadores y el evento r al acuerdo puramente por azar. Por propiedades de la medida de probabilidad, el numerador de κ corresponde efectivamente a la probabilidad de que ambos clasificadores lleguen a una misma conclusión "sin recurrir" al azar.

Si se considera que uno de estos examinadores es precisamente la ground truth presente en los datos este análisis se puede enriquecer aún mas. En términos de objetivos, se espera que un clasificador se comporte en la forma en que lo hace el "oráculo" que proporciona la ground truth, es decir que $\mathbb{P}(a) = 1$, en cuyo caso $\kappa = 1$. Por otro lado, también se espera que nuestro modelo tome decisiones en forma "inteligente" por lo que se busca reducir $\mathbb{P}(r)$, evidentemente hasta cero es el ideal. En cualquier caso el valor óptimo de esta métrica es 1, en el cual la matriz de confusión es diagonal.

Más aún, cabe considerar que tanto $\mathbb{P}(a)$ como $\mathbb{P}(r)$ se obtienen en forma empírica. Luego, si se considera que uno de los examinadores proporciona las clases reales de los datos, es posible escribir ambas probabilidades estimadas en términos de TP, FP, TN y FN:

Claramente, $\mathbb{P}(a)$ corresponde a la *Accuracy* del modelo, pues corresponde a la proporción de aciertos, es decir:

$$\mathbb{P}(a) = Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$

Por otro lado, típicamente la probabilidad $\mathbb{P}(r)$ se modela como:

$$\mathbb{P}(r) = \mathbb{P}_A(Si) * \mathbb{P}_B(Si) + \mathbb{P}_A(No) * \mathbb{P}_B(No) \tag{8}$$

donde $\mathbb{P}i(X)$ denota la probabilidad de que el examinador i determine X. Cuando el examinador B corresponde a la ground truth sus probabilidades estimadas corresponden a:

$$\mathbb{P}_B(Si) = \frac{TP}{TP + TN}, \quad \mathbb{P}_B(No) = \frac{TN}{TP + TN} \tag{9}$$

- **Nota**: Esto corresponde nada más que a la distribución de clases en los datos. Consecuencia de esto, la métrica dada por κ incorpora naturalmente una medida del desbalance en los datos.

mientras que para el examinador A (correspondiente al modelo evaluado):

$$\mathbb{P}_A(Si) = \frac{TP + FP}{TP + FP + TN + FN}, \quad \mathbb{P}_A(No) = \frac{TN + FN}{TP + FP + TN + FN} \tag{10}$$

3. Descripción de Algoritmos

En la presente sección se presenta una breve descripción de los distintos algoritmos de *Machine Learning* utilizados.

3.1. Naïve Bayes

El algoritmo naïve Bayes corresponde a un modelo probabilístico de inferencia basado en el teorema de Bayes bajo la fuerte hipótesis (naïve) de independencia entre las características.

Para el caso de clasificación, dado un vector de características $x = (x_1, x_2, ..., x_n)$, se busca determinar la probabilidad a posterior $p(C_k|x_1, ..., x_n)$ para cada clase C_k, $k = \{1, ..., K\}$, donde por teorema de Bayes, la probabilidad a posteriori puede expresarse como:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} \tag{11}$$

con $p(x|C_k)$ la verosimilitud, $p(C_k)$ la distribución a priori y $p(x)$ la evidencia correspondiente a una constante de normalización.

Utilizando las propiedades de probabilidades condicionales se obtiene que $p(x|C_k)p(C_k) = p(C_k, x_1, ..., x_n)$ y utilizando la propiedad de regla de la cadena en probabilidades la expresión anterior resulta:

$$p(C_k, x_1, ..., x_n) = p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k) \cdots p(x_{n-1}|x_n, C_k)p(x_n|C_k)p(C_k) \tag{12}$$

Utilizando la hipótesis de independencia condicional (naïve) de las características con respecto a las clases, la expresión anterior resulta:

$$p(C_k, x_1, ..., x_n) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{13}$$

Luego, reemplazando en la ecuación (11) y recordando que la evidencia $p(x)$ corresponde a una constante, la distribución a posteriori resulta:

$$p(C_k|x) \propto p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{14}$$

Finalmente, la clasificación estimada $\hat{y}$ corresponde a la estimación máximo a posteriori (MAP), determinada por:

$$\hat{y} = \arg\max_k p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{15}$$

donde la distribución de verosimilitud $p(x_i|C_k)$ es un criterio de diseño de acuerdo al problema. Ejemplos de distribuciones comunes para la verosimilitud son la distribución Gaussiana, Bernoulli o multinomial, entre otras.

3.2. Support Vector Machine

La máquina de soporte vectorial o *support vector machine* (SVM) [1] busca dividir el espacio de un conjunto de datos de dos clases (extensible a n clases) mediante un hiperplano separador $w \cdot x + b = 0$ que maximice el margen $\omega/||\omega||$, es decir, la distancia entre los vectores soporte de ambas clases x_+ y x_-, los cuales satisfacen $w \cdot x_+ + b = 1$ y $w \cdot x_- + b = -1$, respectivamente, siendo las muestras más cercanas al hiperplano separador de cada clase. De este modo, el problema de optimización a resolver puede plantearse de la siguiente forma:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}||w||^2 + c\sum_{i=1}^{N}\xi_i \\ \text{s.a} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i,\ i = 1, \ldots, N \end{aligned} \tag{16}$$

donde x_i corresponde a muestra, $y_i \in \{-1, 1\}$ su clase, ω y b los parámetros y el sesgo del hiperplano separador, respectivamente, ξ_i las variables de holgura, y c el parámetro de tolerancia. Las variables de holgura ξ_i otorgan flexibilidad al hiperplano óptimo, permitiendo que clasifique erróneamente ciertas muestras para evitar el sobreajuste a *outliers* que mermen el desempeño global de clasificación, mientras que el parámetro de tolerancia c determina la suavidad del hiperplano obtenido.

Para conjuntos de datos linealmente no separables, se procede a aumentar la dimensionalidad de los datos mediante una función $\phi : X \mapsto \mathbb{R}^D$, con el objetivo de que en el nuevo espacio, las muestras sean linealmente separables, permitiendo su clasificación mediante SVM. De este modo, reemplazando las muestras x_i por su mapeo de alta dimensionalidad $\phi(x_i)$ en (16), el problema de optimización dual resulta:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\alpha_i\alpha_j y_i y_j \langle\phi(x_i), \phi(x_j)\rangle \\ \text{s.a} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq c \end{aligned} \tag{17}$$

Se observa en (17) que el único término donde aparece $\phi(x)$ es en el producto punto $\langle\phi(x_i), \phi(x_j)\rangle$, el cual corresponde al Kernel $K(x_i, x_j)$. Por tanto, sólo basta escoger una función mercer Kernel $K(x_i, x_j)$ (simétrico y positivo definido) para resolver el problema de clasificación mediante SVM en altas dimensiones, lo cual se conoce como el truco del Kernel. Entre las funciones mercer Kernel más comunes se encuentra el Kernel Gaussiano, el cual se define como:

$$K_{gaussian}(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) \tag{18}$$

con σ el parámetro de ancho de banda del Kernel.

3.3. K-Nearest Neighbors

El algoritmo *k-nearest neighbors* corresponde a un método no paramétrico de clasificación y regresión en base a la distancia entre los k vecinos más cercanos de un punto en el espacio de características.

Considerando el caso de clasificación, sean $\{(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)\} \in \mathbb{R}^d \times \{1, 2, ..., m\}$ un conjunto de entrenamiento compuesto por un dato en el espacio de características y su etiqueta, y dada una norma $||\cdot||$ en $\mathbb{R}^d$, el algoritmo *k-nearest neighbors* computa la distancia ordenada de un nuevo dato x a las muestras de entrenamiento $||X_{(1)} - x|| \leq ||X_{(2)} - x|| \leq ... \leq ||X_{(n)} - x||$, retornando un valor de pertenencia a cada clase de acuerdo a la votación de los k vecinos más cercanos $||X_{(1)} - x|| \leq ... \leq ||X_{(k)} - x||$, donde el nuevo dato x será asignado a la clase más frecuente entre los vecinos, con la cantidad de vecinos k y la norma $||\cdot||$ parámetros de diseño.

Un caso particular del algoritmo *k-nearest neighbors* se tiene al seleccionar $k = 1$, donde un nuevo dato x es clasificado de acuerdo a la clase de su vecino más cercano en el espacio de características $C_n^{1nn} = Y_{(1)}$, garantizando una tasa de error menor o igual al doble del mínimo error alcanzable dada la distribución de los datos (tasa de error Bayesiana) cuando la cantidad de datos de entrenamiento tiende a infinito.

3.4. Random Forest

El algoritmo *random forest* corresponde a un algoritmo de aprendizaje supervisado de ensamblaje de un conjunto de árboles de decisión mediante el método *bagging*.

El método de *bagging* (*bootstrap aggregating*) consiste en la generación de m modelos independientes entrenados con un sub-conjunto obtenido mediante muestreo aleatorio con reposición del conjunto de datos original, donde la inferencia corresponde a un promedio (regresión) o moda (clasificación) de cada uno de estos modelos, reduciendo el error de generalización y el sobreajuste.

En particular, el algoritmo *random forest* entrena B árboles de decisión mediante un sub-conjunto aleatorio de características con reposición (*feature bagging*), donde usualmente, dadas q características, cada árbol decisión B_i es entrenado con $\sqrt{q}$ características.

3.5. Perceptrón Multicapa

El perceptrón multicapa o *multilayer perceptron* (MLP) [2] es un tipo de red neural artificial que procesa un vector de entrada x a través de múltiples capas l, las cuales multiplican el vector de entrada por una matriz de pesos W_l correspondiente a cada capa, adicionando un respectivo término de sesgo b_l. La salida de cada capa es procesada por una función no lineal conocida como función de activación (ej: Sigmoide (σ), tanh, ReLU, etc.) cuyo resultado es sucesivamente utilizado como entrada de la capa siguiente hasta alcanzar la última capa, llamada capa de salida. De este modo, la salida $\hat{y}$ del perceptrón multicapa puede ser expresado como:

$$\hat{y} = \sigma(\cdots\sigma(\sigma(xW_0 + b_0)W_1 + b_1)\cdots)W_l + b_l \tag{19}$$

con W_i y b_i la matriz de pesos y el término de sesgo correspondiente a la i-ésima capa y σ la función de activación sigmoide.

El perceptrón multicapa utiliza una técnica de aprendizaje supervisado llamada *back-propagation* en su etapa de entrenamiento, es decir, dado un conjunto de datos de entrenamiento x con su respectiva salida deseada y, el modelo modifica sus parámetros mediante la propagación del error de predicción determinado por una función de pérdida L entre la salida $\hat{y}$ predicha por el modelo y la salida deseada y.

Dependiendo de la función de pérdida escogida, el perceptrón multicapa es utilizado típicamente para tareas de clasificación o regresión, y en virtud del *Teorema de Aproximación Universal*, las funciones de activación no lineales aplicadas entre capas y el gran número de parámetros del modelo permiten representar una amplia variedad de funciones que se ajusten a los datos con precisión arbitraria.

3.6. Red Neuronal Recurrente

La red neuronal recurrentes o *recurrent neural network* (RNN) es un tipo de red neuronal cuya arquitectura está especialmente diseñada para procesar datos de estructura de series de tiempo gracias a su capacidad de almacenar memoria a través de propagación de estados en el tiempo.

Sean θ los parámetros de la red neuronal, x_t y h_t los datos de entrada y el estado en el tiempo t, respectivamente, la red neuronal recurrente puede expresarse mediante la siguiente función recurrente:

$$h_t = f(x_t, h_{t-1}, \theta) \tag{20}$$

Una de las arquitecturas más simples de redes neuronales recurrentes es la Elman RNN [3], la cual simplemente multiplica matrices de pesos a las entradas y a los estados adicionando un término de sesgo:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{21}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{22}$$

donde W_h, U_h y b_h corresponden a las matrices de pesos relativas a la entrada, el estado anterior y el término de sesgo para cada tiempo t, W_y y b_y corresponden a la matriz de pesos y el término de sesgo de la salida, y σ_h y σ_y corresponden a funciones de activación arbitrarias.

Una de las principales limitaciones de esta arquitectura es su incapacidad de procesar largas dependencias temporales entre los datos siendo, en este escenario, inviable su entrenamiento debido a fenómenos de desvanecimiento y explosión del gradiente del error. Para lidiar con este problema se han propuesto múltiples arquitecturas las cuales añaden compuertas que facilitan el flujo directo de información a través del tiempo. Una de las arquitecturas más famosas es la llamada *Long Short-Term Memory* (LSTM) [4], la cual se compone de múltiples compuertas con matrices de pesos W y sesgos b independientes, las cuales determinan la información debe ser utilizada para actualizar los estados de la red y la información que debe ser desechada en el proceso. Matemáticamente, la red neuronal recurrente LSTM queda determinada por las siguientes compuertas:

$$i_t = \sigma\big(W_i x_t + U_i h_{t-1} + b_i\big) \tag{23}$$

$$f_t = \sigma\big(W_f x_t + U_f h_{t-1} + b_f\big) \tag{24}$$

$$o_t = \sigma\big(W_o x_t + U_o h_{t-1} + b_o\big) \tag{25}$$

$$\tilde{C}_t = \tanh\left(W_g x_t + U_g h_{t-1} + b_c\right) \tag{26}$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \tag{27}$$

$$h_t = \tanh(C_t) * o_t \tag{28}$$

con h_{t-1}, C_{t-1}, los estados ocultos y de celda en el tiempo $t-1$, x_t la entrada en el tiempo t, i_t, f_t y o_t las compuertas de entrada, olvido y salida, respectivamente, y $\tilde{C}_t$, C_t y h_t, el estado de celda propuesto, el estado de celda final y la estado oculto final en el tiempo t.

3.6.1. ELMo

El modelo ELMo (*Embeddings from Language Models*) [5] corresponde a un modelo de neuronal profundo de representación contextualizada de palabras basado en redes neuronales recurrentes bidireccionales capaz de incorporar características complejas del uso de palabras, tales como la sintaxis o la semántica, así como también, información contextual de las mismas.

Sea una secuencia de N tokens $(t_1, t_2, ..., t_N)$ y x_t^{LM} una representación del modelo de lenguaje independiente del contexto de un token t, el modelo ELMo procesa la secuencia de representaciones x_t^{LM} mediante L capas de una red LSTM obteniendo representaciones $\vec{h}_{k,j}^{LM}$ dependientes del contexto para cada posición de la secuencia y capa $j = \{1, ..., L\}$ de la red, donde la salida de la última capa $\vec{h}_{k,L}$ es utilizada para predecir el siguiente token de la secuencia t_{k+1} utilizando la función *softmax*. De este modo, se obtiene una representación $\vec{h}_{k,j}^{LM}$ de un token t_k condicionado a los tokens precedentes $(t_1, t_2, ..., t_{k-1})$, y de manera análoga, es posible obtener una representación $\overleftarrow{h}_{k,j}^{LM}$ del mismo token t_k, pero condicionado a los tokens posteriores $(t_{k+1}, t_{k+2}, ..., t_N)$, obteniendo una representación bidireccional capaz de maximizar la verosimilitud de ambas direcciones:

$$\sum_{k=1}^{N} \log(p(t_k|t_1, t_2, ..., t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log(p(t_k|t_{k+1}, t_{k+2}, ..., t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \tag{29}$$

con t_k los distintos tokens de la secuencia, Θ_x los parámetros del modelo de representación independiente de contexto de los tokens, $\vec{\Theta}_{LSTM}$ y $\overleftarrow{\Theta}_{LSTM}$ los modelos LSTM de representación contextual de los tokens precedente y posterior, respectivamente, y Θ_s los parámetros de una capa *softmax*.

Las distintas representaciones intermedias de los modelos de lenguaje bidireccional se combinan dependiendo de la aplicación. Para cada token t_k, el modelo bidireccional de lenguaje de L capas computa un conjunto de $2L + 1$ representaciones:

$$R_k = \{x_k^{LM}, \vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, ..., L\} = \{h_{k,j}^{LM} | j = 1, ..., L\} \tag{30}$$

Para obtener la representación final, se colapsan las representaciones de todas las capas en R en un sólo vector $\text{ELMo}_k = E(R_k; \Theta_e)$. El caso más simple consiste en seleccionar sólo las últimas de modelo $E(R_k) = h_{k,L}^{LM}$, mientras que en general, se escogerá una versión ponderada de cada representación:

$$\text{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM} \tag{31}$$

donde s^{task} corresponde al vector de pesos normalizados mediante *softmax* y γ^{task} un factor de ponderación de las distintas representaciones.

3.7. Transformer

El modelo *transformer* [6] propone utilizar únicamente mecanismos de atención para modelar la dependencia temporal de una secuencia, prescindiendo completamente del modelo neuronal recurrente, permitiendo la paralelización del mismo, disminuyendo considerablemente los tiempos computacionales de procesamiento y mejorando, a la vez, su desempeño frente a distintas tareas.

El modelo propuesto sigue la estructura encoder-decoder, donde el encoder mapea una secuencia de entrada $(x_1, ..., x_n)$ a una representación continua $z = (z_1, ..., z_n)$, mientras que el decoder, dada la representación intermedia z, genera una secuencia de salida $(y_1, ..., y_m)$ de los símbolos de manera autoregresiva, es decir, utiliza todos los símbolos previamente generados para generar el siguiente, donde el encoder y decoder del modelo *transformer* se compone de un apilamiento de mecanismos de auto-atención, atención punto a punto, y capas *fully-connected*.

La función de atención corresponde a un mapeo de una *query* y con conjunto de valores pares *key-value* a una salida. Dado los vectores *query*, *key* y *value*, la salida corresponde a la suma ponderada de los elementos del vector *value*, cuyos pesos son computados mediante una función de compatibilidad entre el vector *query* y el vector *key*.

Sean Q, K, V matrices correspondientes a un conjunto de *queries*, *keys* y *values* de dimensión d_k, d_k y d_v, respectivamente, se define la función de atención producto punto escalda como:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right)V \tag{32}$$

En vez de ejecutar una sola función de atención sobre *queries*, *keys* y *values* de dimensión d_{model}, se propone proyectar dichos vectores h veces mediante proyecciones lineales de dimensión d_k, d_k y d_v, respectivamente, donde cada proyección corresponde a aplicación de la función de atención en paralelo, retornando una salida de dimensión d_{model}, las cuales son concatenadas y proyectadas nuevamente hasta obtener los valores finales. Esta modificación de la función de atención se denomina *multi-head attention*, y queda definida como:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{33}$$

con $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ y $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ las matrices de proyección.

De este modo, el modelo *transformer* aplica las función *multi-head attention* en capas de atención encoder-decoder, donde las *queries* corresponden a la salida de la capa previa del decoder, mientras que los valores *keys* y *values* corresponden a la salida del encoder, permitiendo que cada posición en el decoder sea capaz de atender todas las posiciones de la secuencia de entrada.

Adicionalmente, cada capa del enconder computa la función *multi-head attention* cuyas *queries*, *keys* y *values* vienen de un mismo lugar (auto-atención) correspondiente a la salida de la capa previa del encoder, permitiendo que a cada posición del encoder atender a todas las posiciones de la capa previa del mismo. De modo similar, el decoder aplica el mismo mecanismo de auto-atención restringiendo el flujo de información hacia la izquierda mediante el enmascaramiento de los valores de entrada correspondientes a conexiones que violen la propiedad auto-regresiva.

Finalmente, la salida de capa de atención en el encoder y decoder pasa por una red *fully-connected* de dos transformaciones lineales con función de activación *ReLU* aplicado de manera

separada e idéntica a cada posición de la secuencia y los tokens de entrada del modelo son transformados a un *embedding* de dimensión d_{model} fija, donde la salida del modelo corresponden a una transformación lineal con función $softmax$ de la última capa del decoder.

En la figura 2 se presenta un diagrama de la arquitectura del modelo *transformer* anteriormente descrita, donde N_x corresponde a una capa del encoder o decoder.

3.7.1. BERT

El modelo BERT (*Bidirectional Encoder Representations from Transformers*) [7] corresponde a un modelo *transformer* de codificación bi-direccional de arquitectura multipropósito.

La restricción de unidireccionalidad del modelo *transformer* (propiedad auto-regresiva) se relaja mediante un entrenamiento de lenguaje enmascarado del modelo (*masked-language model*), en el cual se enmascara aleatoriamente algún token de la secuencia de entrada, con el objetivo de predecir la id del token oculto. Adicionalmente al *masked-language model*, el modelo BERT se entrena bajo la tarea de predicción de oración posterior (*next sentence prediction*), el cual utiliza conjuntamente representaciones de texto pareadas pre-entrenadas, permitiendo de este modo, obtener representaciones bidirecionales del lenguaje altamente informativas.

El modelo pre-entrenado bajo estas tareas puede ser aplicado a cualquier otra añadiendo capas *fully-connected* sobre alguna representación de salida de interés, entrenadas mediante *fine-tuning*.

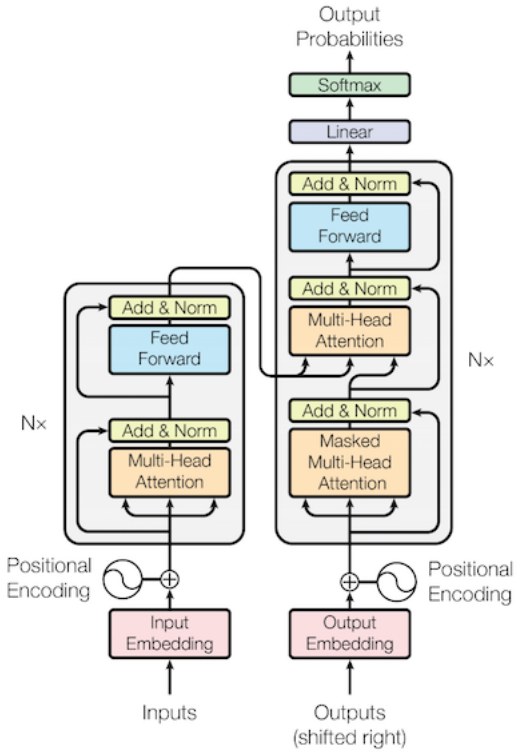

Figura 2: Esquema de arquitectura de modelo *transformer*.

4. Descripción de Representaciones

En esta sección se presentan y fundamentan las representaciones estudiadas durante la confección del presente proyecto.

4.1. Baseline

A modo de *baseline* se presenta un modelo de clasificación basado en vectores de características conformados por dos componentes:

- Unigramas: Dado que dicho modelo base consiste en un clasificador *Naive Bayes*, es imperante considerar una representación basada en unigramas, de modo tal que no se violen los supuestos detrás de este algoritmo de clasificación. Por lo tanto, para vectorizar cada tweet se emplea la transformación `CountVectorizer` nativa de la librería *Sklearn*.
- Contadores de caracteres especiales: Por medio de la transformación `CharsCountTransformer` proporcionada se representa cada tweet según la cantidad de usos de los caracteres "#", "!" y "?". Cada cuenta se almacena en una de las tres componentes que forman dicho vector de representación.

Ambas componentes (una de alta dimensionalidad, dada por el tamaño del Corpus y la cantidad de palabras distintas empleadas en este y otra de tres dimensiones) se concatenan para formar la representación vectorial empleada en este modelo.

Más allá de este *baseline*, se consideraron en este trabajo otras representaciones (tanto en forma complementaria como excluyente), entre las cuales se encuentran:

4.2. N-gramas

Para aprovechar las obvias relaciones semánticas presentes en el lenguaje es posible relajar la condición de independencia condicional adoptada en el punto anterior, pasando de emplear unigramas a usar estructuras de orden superior (n-gramas). Una vectaja que ofrece el extractor `CountVectorizer` es que cuenta con un parámetro para indicar el/los órdenes de n-gramas deseados, pudiendo retornar tanto representaciones en un orden específico (unigramas, bigramas, trigramas, etc. en forma excluyente) como varios órdenes en conjunto, es decir unir las representaciones asociadas a cada orden.

Si bien en el archivo adjunto donde se realizaron las pruebas se muestran resultados asociados a unigramas y brigramas (tanto su uso por separado como en forma simultánea), pruebas utilizando representaciones de hasta un orden $n = 5$ fueron consideradas, siendo descartadas en forma inmediata dado su pobre desempeño. Las pruebas que sí se exponen en dicho archivo son lo suficientemente representativas del comportamiento general del problema frente al aumento del orden de la representación de n-gramas.

4.3. Emojis

Dado el contexto del problema, cobra especial relevancia el uso de emojis por diversas razones:

- El Corpus se compone de tweets, los cuales son un medio comunicacional que, en forma natural, no se limita al uso de palabras propiamente tal, sino que también recurre a siglas, símbolos ASCII y otros símbolos no reconocidos dentro de dicho código como es el caso de los emojis, los cuales tienen un inherente contenido emocional y semántico.
- Dado que estos no son reconocibles para ciertas herramientas, por ejemplo la misma codificación ASCII, el manejo nativo de `Strings` en Python y, más importantemente en este caso, el extractor empleado para este trabajo `CountVectorizer`, al emplearse este en la representación de la información todo contenido presente en los emojis presentes en los tweets se perdería.

Para esta tarea se emplea el módulo `emoji` y una implementación basada en la realizada en `twitter-sentiment-analysis` [8]. En base a una lista de diversos emojis asociados a distintas emociones se establece un token correspondiente a emociones positivas y negativas, por el cual los emojis listados son reemplazados en el tweet. El vector de características corresponde entonces a un contador de las instancias de cada uno de estos tokens. Las cuentas debieran tener una correlación con la emoción general del tweet, tanto a nivel de categoría como a nivel de intensidad.

4.4. Baseline Modificado

Siguiendo la misma filosofía del contador de caracteres especiales utilizado en el *baseline*, al observar el conjunto de entrenamiento se establece que se omitieron en el extractor `CharsCountTransformer` dos caracteres que podrían ser informativos a la hora de clasificar, los cuales corresponden a "..." y "@". Particularmente el segundo se estima que puede ser muy informativo, pues se puede asociar o hipotizar sobre la relación entre las menciones a otros usuarios (ya sean personas u organizaciones) a emociones de distinto espectro e intensidad, como por ejemplo hacer un reclamo, invitar a otros usuarios a compartir algo o informarles de alguna situación, etc. El uso del punto suspensivo puede asociarse también al énfasis sintáctico y, por ende, a la propia emoción detrás de lo enunciado.

Por estas razones se incorpora una versión modificada del mencionado extractor, denominado en el programa adjunto por `ModifiedCharsCountTransformer`.

4.5. Lexicons

Tal como se comenta en las recomendaciones para el presente proyecto, el uso de Lexicons diseñados precisamente para la clasificación de emociones puede ser una herramienta con la cual llegar a mejoras en la representación y extracción de información. En particular, se emplea el Lexicon diseñado por Hu & Liu [9]. Si bien el mismo Liu alude a que el uso de "*opinion words*" no implica en forma categórica que la oración que las contiene corresponda a determinada emoción u intensidad, sí puede ser un factor influyente, por lo que en base a este lexicon se establecen contadores, en forma similar a como se hizo con el tratamiento de los emojis.

4.6. Tramiento del Énfasis

En línea con algunas de las representaciones anteriores, en las cuales se busca establecer relaciones con las categorías a explorar y la expresividad propia de algunos canales digitales como es el caso de twitter, se formula otro contador de estas expresiones para preservar un poco de su connotación e implicancia semántica que se pierde al normalizarlas por medio de `CountVectorizer`. Más precisamente, se propone una representación para contabilizar el uso de palabras exclusivamente en mayúsculas (que se pueden asociar a la intensidad subyacente en el mensaje) y de las palabras con elongación de caracteres, pues dicha elongación puede asociarse también al lenguaje cotidiano e intención detrás del mensaje, más precisamente a la propia emoción e incluso al uso del sarcasmo.

4.7. Embeddings

Adicionalmente, se decidió explorar la incorporación de *embeddings* mediante modelos neuronales profundos. Para ello se investigó en la literatura y se decidió utilizar representaciones embebidas bidireccionales de oraciones capaces de incorporar características complejas en las mismas, tales como la sintaxis, la semántica, la información contextual, etc., mediante dos distintos modelos representantes de los enfoques neural recurrente (ELMo) y de atención (BERT), respectivamente.

La representación obtenida mediante el modelo ELMo corresponde a un vector de dimensión 1024, correspondiente al promedio simple de las representaciones de las últimas capas $R_k = \{x_k^{LM}, \vec{h}_{k,L}^{LM}, \overleftarrow{h}_{k,L}^{LM}\}$ descritos en la ecuación (30), mientras que la representación obtenida mediante el modelo BERT corresponde a un vector de dimensión 1024 correspondiente a la salida de clasificación [CLS] de la última de transformación. De acuerdo a las recomendaciones de los autores de los trabajos [5] y [7], se utilizó el modelo ELMo como un simple extractor de características, mientras que al modelo BERT se le realizó un *fine-tuning* en el conjunto de datos de tweets utilizado.

Para utilizar los modelos anteriormente mencionados, se debió pre-procesar cada tweet del conjunto de datos, el cual incluye eliminación de url's, sujetos, números y puntuaciones, conversión de emojis a texto, transformación de contracciones a palabras completas, traducción de acrónimos, conversión a caracteres minúsculos y separación de palabras.

5. Experimentos

En esta sección se presenta el método de evaluación de las distintas representaciones y modelos considerados, junto a un análisis comparativo de los resultados obtenidos. Para mayor detalle de los resultados de cada experimento, revisar archivos `ipynb` adjuntos.

A grandes rasgos la filosofía detrás del proceso de evaluación y selección es una incremental, en base a la cual se va añadiendo complejidad en la medida que el modelo exhibe progresivamente mejores resultados sobre esta misma complejidad. En otras palabras, el proceso general consiste en evaluar un modelo simple, añadir un grado de complejidad (por ejemplo incluir nuevos *features* al vector de características que representa los datos), evaluar la ganancia relativa. En caso de obtener una mejora consistente y relativamente considerable se considera el grado de complejidad probado como el nuevo "*benchmark*" a superar y al cual añadir complejidad en las pruebas posteriores; en caso contrario se descarta el grado de complejidad probado.

La finalidad de esta metodología es evitar un proceso combinatorial de búsqueda, en el cual todas las combinaciones de representaciones y de algoritmos son probadas, lo cual implicaría grandes tiempos de cómputo, y el hecho de que hasta que no se realicen todas las pruebas no se tiene noción alguna del aporte o incremento en *performance* que conlleva la adición de cada *feature* o cambio en el algoritmo de clasificación.

Además, este esquema se puede justificar en el hecho de que cada una de las representaciones presentadas (salvo por el uso de modelos de *Embedding* en reemplazo del extractor `CountVectorizer`) puede interpretarse como independiente de las demás, dada la clase a la que pertenecen. Por ejemplo, incorporar palabras escritas en mayúsculas y el uso de emojis no tienen una correlación estadística tan fuerte, dada la intensidad de la emoción.

De esta forma, la metodología general seguida consiste en probar las distintas representaciones (basadas en `CountVectorizer`), primero para identificar aquellas que conllevan una mejora por sí mismas para luego combinarlas en forma incremental hasta encontrar la "óptima". Sobre esta representación se prueban distintos clasificadores. De esta forma se busca encontrar aquellos que ofrezcan un desempeño superior a los demás, al ser los que mejor explotan la información presente en los datos de entrenamiento dada la representación escogida.

Durante la primera etapa se considera una semilla para formar la partición del *dataset*, de modo tal que las pruebas sean comparables, puesto que al tratarse de pruebas sobre la extracción de información se considera pertinente que la información a extraer sea la misma. De esta manera se podrá establecer qué extractores son capaces de extraer la mayor cantidad de información y tener certeza de que la información contenida en primer lugar era la misma a lo largo de las distintas pruebas.

Cabe destacar que en un principio no se tenía contemplado el uso de técnicas de *embedding*, pero en vista de los resultados obtenidos mediante la metodología descrita se optó por recurrir a estas, particularmente a los enfoques basados en redes recurrentes y en *transformers*, realizando experimentos de desempeño sobre los mejores clasificadores encontrados en las pruebas previas, escogiendo la representación de mejor desempeño para ser incorporada a las características previamente extraídas, debido a que ambas representaciones tratan de capturar la información sintáctica, semántica y contextual de las oraciones, por tanto, incorporar ambas resultaría redundante.

5.1. Balance de clases y *Sanity Check*

La primera prueba a considerar es tratar el desbalance de clases, puesto que de no hacerlo el clasificador se puede sesgar, particularmente hacia la clase mayoritaria, pudiendo perjudicar el desempeño general del mismo. Más precisamente, al contar con pocos datos que además presentan un desbalance, al particionar los datos para formar los conjuntos de entrenamiento y validación es muy fácil que ambos conjuntos sean poco representativos del fenómeno. Por esta misma razón, el método de *Sklearn* diseñado para realizar esta partición, `train_test_split`, cuenta con un parámetro denominado `stratify`, el cual establece que al momento de realizarse la partición se preserven las frecuencias relativas de las clases.

Haciendo esto se obtiene un incremento considerable en la performance. Más precisamente, se obtiene una mejora de aproximadamente 3 % en la métrica AUC, 1 % en Kappa y obteniendo resultados similares en *Accuracy*. Si bien estos resultados pudieran parecer marginales, se comprobó a lo largo del proyecto que los incrementos de desempeño suelen ser mucho menores. Más aún, las dos primeras métricas son precisamente las más complicadas de mejorar. De hecho cuando se incrementan suele empeorar la *Accuracy*, por lo que mejorarlas en forma tan importante sin dañar en esta última es un factor importante a considerar

En vista de lo anterior, se comprueban los beneficios de estratificar los conjuntos de entrenamiento y validación, por lo que para las pruebas posteriores se incorpora esta medida.

Por otro lado, la metodología descrita se sustenta en el hecho de que las características añadidas a la representación obtenida por medio de `CountVectorizer` satisfagan los supuestos del modelo de clasificación *Naive Bayes* y que efectivamente al añadirlas se obtenga una mejora en la generalización del modelo. Por esta razón se realiza una primera prueba donde el mismo clasificador del *baseline* es entrenado sin considerar al extractor `CharsCountTransformer`, es decir solamente con la vectorización de los tweets por medio de `CountVectorizer`.

Si bien la diferencia en desempeño entre el modelo *baseline* y el que no considera los contadores dados por `CharsCountTransformer` es considerablemente baja, siendo la variación más importante la obtenida en la métrica Kappa, la cual es precisamente la más volátil, y del orden de 0,5 %; el modelo original es consistente en todas las métricas, en el sentido que ofrece un mejor desempeño en todas estas.

De este modo se constata que efectivamente el añadir nuevas características al vector de representación permite mejorar el modelo obtenido por medio de *Naive Bayes*, justificando las pruebas presentadas a continuación. La primera de estas pruebas e corresponde a la extracción de características basada en emojis.

5.2. Experimentos con Representación basada en Emojis

Esta prueba consiste en comparar la *performance* entre el modelo obtenido en el punto anterior, consistente en un extractor de *features* que emplea `CountVectorizer` y `CharsCountTransformer` y un clasificador *Naive Bayes* y uno que reemplace en el vector de representación de este último la información extraída por medio de `CharsCountTransformer` por la de `EmojiHandler`, que es la clase que realiza las transformaciones mencionadas en la sección 4.3.

En forma inversa a como ocurrió en el experimento anterior, en este caso los resultados son

consistentemente mejores al añadir el manejo de emojis por medio de contares. Particularmente la métrica Kappa es la que muestra una mayor ganancia, por lo que al ser una métrica propensa a disminuir debido a su volatilidad se concluye que las características extraídas a partir de emojis son muy informativas.

5.3. Experimentos con `CharsCountTransformer` Modificado

Como se mencionó en la sección 4.4, la idea de este este experimento es enriquecer el contador de caracteres especiales proporcionado en el *baseline*.

Teniendo dicho experimento como referencia, al incorporar las modificaciones a `CharsCountTransformer` se obtiene una mejora muy sutil en cuanto al área bajo la curva ROC, pero perjudicando a las otras dos.

En síntesis, la ganancia lograda por las modificaciones realizadas a `CharsCountTransformer` son nimias cuando son positivas y considerables cuando son negativas.

5.4. Experimentos con Lexicon

En esta sección se estudia el desempeño del clasificador *Naive Bayes* considerando como *features* las obtenidas por medio de `CountVectorizer` y el extractor de la sección 4.5.

Cabe destacar que este lexicon se encuentra disponible en la librería `nltk`, donde se implementa con un tokenizador adaptado para tweets, donde un factor importante es la preservación del *case*, es decir el uso de mayúsculas o convertir todo a minúsculas. Por esta razón se prueba el extractor basado en contadores considerando dicha conversión, así como los efectos de tomar los datos "crudos".

A grandes rasgos, en términos de área bajo la curva ROC se obtienen resultados similares a los experimentos anteriores, así como de *accuracy* (aunque cabe destacar que estos son levemente menores al *baseline*). El factor más afectado es la métrica Kappa, exhibiéndose los peores resultados obtenidos hasta este punto. Se aprecia además un leve *trade-off* entre Kappa y las otras dos métricas, donde al preservar el uso de mayúsculas el aumento en la primera repercute negativamente en las otras dos. Sin embargo las diferencias son ínfimas. Esto último es interesante puesto que cabría esperar un efecto mayor pues al preservar las mayúsculas debería perderse información, puesto que los contadores se realizan sobre el listado de palabras del lexicon formulado en minúsculas. Esto indicaría que dentro del *dataset* empleado el uso de mayúsculas no es tan considerable ni relevante.

5.5. Experimentos con `EmphasisHandler`

En esta sección se estudia el efecto de reemplazar en el *baseline* el extractor `CharsCountVectorizer` por el `EmphasisHandler` presentado en 4.6. En este caso todas las métricas de desempeño disminuyen en forma considerable respecto al modelo original, salvo por la *accuracy*, la cual empeora en una menor medida (0,2 % en promedio).

Tal como en la sección anterior, esto sugiere que el uso de mayúsculas prácticamente es irrelevante en el presente problema, lo cual también pareciera ocurrir para la duplicación de caracteres.

Al realizar una inspección de los datos correspondientes a cada sentimiento se observa que, efectivamente, la cantidad de tweets en los cuales se incurre al uso de mayúsculas para escribir palabras completas y duplicar reiteradamente caracteres es nula para efectos prácticos. En el archivo adjunto se muestran dos ejemplos de tweets donde los contadores implementados en `EmphasisHandler` tienen utilidad y algún posible impacto positivo, debido al claro significado semántico que tiene la forma en que se escribieron las palabras.

De la inspección de los *datasets* se observa además que, de entre las muy pocas instancias en que se presentan palabras escritas completamente en mayúsculas, la mayoría corresponder a siglas asociadas a entidades, como ESPN. Evidentemente estos casos poseen información muy pobre respecto a la intensidad del sentimiento estudiado por lo que se concluye que la relevancia de las características estudiadas en este experimento es muy pobre para la tarea y datos en cuestión.

5.6. Combinación de *Features*

Ya teniendo una noción de las repercusiones en el desempeño de las características diseñadas se procede a realizar las pruebas incrementales, combinándolas para así extraer la mayor cantidad de información posible.

Primeramente, se añade al modelo *baseline* el extractor de características `EmojiHandler`. En este caso se obtienen resultados prácticamente idénticos en promedio a los exhibidos en los experimentos de la sección 5.2.

Por otro lado, si en lugar de `CharsCountTransformer` se considera su versión modificada para este experimento, se tiene una leve ganancia en términos de AUC, pero una fuerte disminución en Kappa, por lo que de nueva cuenta, aún con la información añadida de los emojis, la ganancia asociada a los contadores de puntos suspensivos y del caracter "@" es cuestionable. Pese a esto, se seguirán considerando para purbas posteriores y así no descartar este extractor de características en forma apresurada.

En síntesis, el extractor basado en emojis pruba ser una herramienta importante para la tarea estudiada, al ser consistente en cuanto a las mejoras que ofrece.

Toca añadir a la mezcla el el extractor de *features* basado en el lexicon de Hu & Liu. Nuevamente y por las mismas razones de no descartarlo en forma temprana se considera el efecto de su tokenizador.

Primeramente, considerando el modelo *baseline* con `EmojiHandler` y el lexicon, los resultados son ambiguos, en el siguiente sentido: tanto permitiendo que el tokenizador convierta todas las palabras a minúsculas como si se preserva la escritura original, se logra alcanzar consistentemente una mayor área bajo la curva lelgando a valores del 68 %, manteniendo los valores de *accuracy*, pero dañando importante mente la métrica Kappa disminuyéndola en casi 1,5 %.

Este efecto es aún mayor si se considera el reemplazo del extractor `CharsCountTransformer` por `ModifiedCharsCountTransformer`, donde se incrementa en un 0,3 % más el resultado anterior en AUC, aumenta la *accuracy en* 1 %, pero incrementando la disminución en Kappa llegando a una pérdida del 2 % respecto al modelo *baseline* con `EmojiHandler`.

Antes de realizar las pruebas añadiendo el extractor `EmphasisHandler` se opta por escoger un "*top* 4" de entre las pruebas realizadas hasta el momento, con la finalidad de reducir la cantidad de combinaciones a testear. Dado el poco impacto positivo presentado por este último extractor

de características, se considera que se puede reducir el número de combinaciones a probar a continuación debido a que es esperable y probable que en muchas de estas combinacines el desempeño sea aún más pobre. Los 4 modelos escogidos para esta prueba son:

- *Naive Bayes* + `EmojiHandler`
- *Baseline* + `EmojiHandler`
- *Baseline* + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)
- *Naive Bayes* + `ModifierCharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

La elección de estos 4 modelos se hace en base a que los dos primeros son los que permiten tener una mayor *accuracy* en general y comparados con los otros experimentos, a la vez que una AUC relativamente alta, sumado a los mayores valores de Kappa. Por su parte, los dos últimos modelos escogidos presentan una pérdida marginal de *accuracy*, pero alcanzando los mayores valores de área bajo la curva ROC, con su consecuente disminución en Kappa. El objetivo es lograr encontrar una combinación que permita balancear en mejor medida los modelos escogidos, los cuales corresponden a los que alcanzan incrementos más importantes respecto al *baseline*.

Al incorporar el extractor `EmphasisHandler` los resultados son concluyentes:

- En el primer caso, se obtiene en promedio una ganancia mínima en cuanto a AUC, perjudicanco importantemente las otras dos medidas de desempeño, es decir que no se obtiene el balance deseado.
- El segundo caso es en cierto sentido contrario. En promedio las dos últimas métricas preservan su desempeño, dañando el área bajo la curva ROC, aumentando en consecuencia el desbalance de *performance*.
- Aún más grave es el resultado del tercer modelo, en el cual todas las medidas de desempeño se ven perjudicadas, obteniéndose claramente un modelo con menor capacidad predictiva.
- El último modelo probado incorporando `EmphasisHandler` es, de entre estas cuatro últimas pruebas, la más favorable. Si bien se perjudica ligeramente la AUC (obteniéndose aún así un 68 %), se logra una recuperación de la métrica Kappa, aportando por lo tanto al balance deseado.

Tras realizar estas cuatro pruebas se determina que el nuevo "*top* 4" consiste en reemplazar solamente el último, al añadir un poco de balance preservando en términos generales los altos desempeños. Por lo tanto dicho "*top* 4" queda como:

- *Naive Bayes* + `EmojiHandler`
- *Baseline* + `EmojiHandler`
- *Baseline* + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

- *Naive Bayes* + `ModifierCharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`) + `EmphasisHandler`

Con esto, se dan por concluidas las pruebas sobre los vectores de características construidos, exceptuando la consideración de n-gramas. Pero para poder incorporar estos en el estudio es necesario redefinir el clasificador, dado que el modelo *Naive Bayes* no está pensado para este tipo de representación.

Dado que en general los clasificadores tipo *random forest* ofrecen una serie de ventajas respecto a otros algoritmos clásicos de *machine learning* (son un ensamble de clasificadores que en conjunto suelen dar un mejor desempeño, son rápidos de entrenar y, al menos en *scikit-learn* cuentan con un parámetro denominado `class_weight` que permite, por ejemplo, ponderar las predicciones por la frecuencia relativa de las clases precisamente para enfrentar el desbalance de datos) se opta por realizar las pruebas con n-gramas empleando este tipo de clasificadores.

Tras estas pruebas, se establece en forma categórica que añadir n-gramas de orden superior repercute fuerte- y perjudicialmente en el desempeño. En el archivo adjunto se muestran los resultados obtenidos al añadir a los modelos anteriores basados solo en unigramas (más los *features* escogidos) una estructura de bigramas y también al reemplazar dichos unigramas por bigramas. En forma reiterada y consistente se observa que la incorporación de bigramas daña la *performance* de los modelos escogidos basados en unigramas; y el impacto de reemplazar dichos unigramas por bigramas es aún más perjudicial.

Se realizaron algunas pruebas incorporando trigramas y n-gramas de orden 4 y 5 de manera similar a la mencionada para bigramas, donde se observó que esta pérdida de desempeño se acentuaba aún más.

Dicho lo anterior y finalmente habiendo escogido las representaciones más útiles (en términos de desempeño) se procede a evaluar el uso de otro tipo de clasificadores distintos a *Naive Bayes* y a los recién testeados *random forest*.

5.7. Experimentos con otros Clasificadores

En la presente sección se estudia el impacto del propio clasificador en la *performance* del modelo, para lo cual se consideran como *features* las siguientes:

- `CountVectorizer` + `EmojiHandler`
- `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler`
- `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)
- `CountVectorizer` + `ModifierCharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`) + `EmphasisHandler`

Los clasificadores a evaluar consisten en Máquina de Soporte Vectorial (*Support Vector Machine*), K-*Nearest Neighbors* y Perceptrón Multicapa.

Para el caso de una máquina de soporte vectorial (*SVM*) entrenada sobre la representación dada por `CountVectorizer` + `EmojiHandler` presenta un evidente peor desempeño, puesto que en promedio todas las métricas se ven disminuidas respecto al *baseline*, obteniéndose valores de AUC por debajo del 66 %, disminuyendo su Kappa en casi 4 % y alcanzando una inédita *accuracy* del 59 %.

No obstante, se aprecia que considerando las otras tres representaciones, por lo demás más complejas, se pueden obtener ganancias marginales de AUC, *accuracies* que rondean el 61 % y en promedio mostrando una Kappa de 0,2, siendo en términos generales los modelos más sólidos obtenidos hasta el momento. Si bien se logran mejoras evidentes y balanceadas en promedio, vale la pena observar lo que ocurre para cada emoción clasificada, puesto que el hecho de obtener un buen desempeño en el promedio sobre las 4 emociones a costa de perjudicar fuertemente una o más de estás es una clara muestra de desbalance, por más que las estadísticas generales mejoren.

Bajo este concepto se aprecia una tendencia clara. No solo los desempeños promediados sobre todas las emociones se ven beneficiados en prácticamente todas las pruebas realizadas con este algoritmo de clasificación, sino que se observan otros patrones como los siguientes: al parecer, en pos de mejorar los desempeños generales sobre las emociones *fear*, *joy* y *sadness* se deteriora en forma importante el desempeño sobre el sentimiento *anger*, llegando por ejemplo a valores de Kappa del 0,33 % de los obtenidos en el *baseline*. Estos resultados se contraponen a la idea de balance buscada.

Prácticamente opuesto es el caso de las pruebas realizadas con un clasificador k-*nearest neighbors* (*KNN*). En estas las pérdidas de desempeño son absolutas y sobre todas las emociones. El resultado de este deterioro es que en todas las pruebas hechas con este tipo de clasificador se obtiene, en promedio, una *Average* AUC menor al 60 %, una Kappa menor a 0,1 y, quizás la disminución más seria en desempeño, una *accuracy* menor a 0,5, lo cual corresponde a una pérdida de 10 % en esta métrica respecto al *baseline*.

Se hace evidente entonces que el uso de clasificadores *KNN* no es uno adecuado para el problema estudiado.

Para el caso de las pruebas realizadas con clasificadores *MLP*, en términos generales se tiene que: en cuanto a la AUC se obtienen resultados similares a los mejores obtenidos por medio de *Naive Bayes*. Respecto a la Kappa los resultados son consistentemente mejores a los obtenidos anteriormente, llegando en promedio a 23,5 %. No obstante, se observa un deterioro en cuanto a la *accuracy*, y en forma generalizada de al rededor del 2 %.

Cabe notar que en el archivo adjunto no se muestran las pruebas realizadas al variar la arquitectura de la red e hiperparámetros de entrenamiento, solo se muestra la configuración elegida, considerada como la más consistente para los *features* empleados.

Si bien los resultados obtenidos por medio de clasificadores *SVM* o *random forest* pudieran parecer rotundamente superiores, el uso de redes *fully-connected* para el presente problema presenta algunas ventajas relativas a estos modelos:

Respecto a las máquinas de soporte vectorial, como ya se discutió anteriormente su buen desempeño es en parte producto de resultados desbalanceados si se compara un sentimiento con otro. En este aspecto las redes *MLP* ofrecen resultados menos dispares entre las distintas emociones, donde además se incrementa notable- y consistentemente la Kappa obtenida para cada emoción.

Al comparar los modelos neuronales estudiados con los experimentos realizados con clasificadres

random forest se concluye que, pese a ser levemente peores en desempeño en cuanto a AUC y *accuracy*, los resultados obtenidos mediante redes *MLP* son comparables a los de *random forest*. Más aún, dado el mencionado *trade-off* entre las distintas métricas de desempeño se concluye que estos modelos son la mejor representación de dicho *trade-off* a la vez que se alcanza una *performance* superior en forma relativa a las demás pruebas realziadas. Es decir, en términos generales se considera que ambos esquemas permiten obtener resultados más favorables y mejor balanceados en comparación a las demás pruebas realizadas; aún así, se percibe que la principal ganancia obtenida por medio de estas redes neuronales es en cuanto a Kappa, mientras que para *random forest* la ganancia es más sustancial en cuanto a AUC y *accuracy*.

Más concisamente, los modelos identificados como los mejores hasta el momento son:

- *Random Forest* con balance de clases + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)
- Red *MLP* (con tres capas ocultas de dimensión 2000, 500 y 100) + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

Una forma de mejorar el balance obtenido por estos últimos modelos es precisamente recurrir a la filosofía detrás de *random forest*: ensambles. Para esto se recurre al `VotingClassifier` implementado en *Sklearn*.

5.8. *Voting Classifier*

Como se comentó anteriormente, en la presente sección se emplea el clasificador `VotingClassifier` para encontrar un ensamble de distintas instancias de los siguientes clasificadores:

- *Random Forest* con balance de clases + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)
- Red *MLP* (con tres capas ocultas de dimensión 2000, 500 y 100) + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

con la finalidad de mejorar el balance obtenido por cada uno entre las métricas de desempeño estudiadas.

Antes de continuar vale la pena discutir sobre el proceso de decisión que realiza el clasificador `VotingClassifier`. En términos simples, este clasificador consiste simplemente en un ensamble, el cual cuenta con un listado de clasificadores los cuales son entrenados en paralelo. La decisión que toma el ensamble consiste en el resultado de un proceso de votación: cada clasificador del listado realiza su propia predicción, por lo que la opción mayoritaria es la retornada por el ensamble `VotingClassifier`. Además, esta implementación de ensambles ofrece dos formas de votación, definidas por el parámetro `voting`, el cual puede tomar los valores `'hard'` y `'soft'`. La primera se trata de una votación basada en la decisión tomada por los clasificadores del ensamble, retornando

la opción mayoritaria. Por otro lado, el segundo método utiliza las probabilidades retornadas por los clasificadores, para cada posible clase se suman las probabilidades asignadas por cada clasificador y se retorna la clase asociada al argumento máximo de esta suma. Dado que el método de *scoring* considerado para este proyecto requiere imperantemente que el clasificador evaluado retorne una probabilidad (por medio del método `predict_proba`) se requiere usar entonces el parámetro `voting = 'soft'`.

Las pruebas realizadas consisten en cuatro modelos `VotingClassifier`, basados en las últimas configuraciones mencionadas para *random forest* y *MLP*. Estos modelos son:

- Clasificador `VotingClassifier` con 3 clasificadores *random forest* y 2 *MLP*
- Clasificador `VotingClassifier` con 3 clasificadores *random forest* y 1 *MLP*
- Clasificador `VotingClassifier` con 3 clasificadores *random forest*
- Clasificador `VotingClassifier` con 5 clasificadores *random forest*

El primer ensamble estudiado, si bien preserva la *performance* del *random forest* en cuando a AUC y la Kappa obtenida por el modelo neuronal, deteriora la *accuracy* por debajo del 60 %.

Al emplear solo una red *MLP* en conjunto con tres clasificadores *random forest* se obtiene una leve mejora en cuanto a AUC llegando a 0,696, se obtiene una Kappa de 0,236, que es 0,005 menor a la obtenida solo por la red pero casi 0,03 mayor a la obtenida por solo un modelo *random forest*. A su vez, la *accuracy* supera en gran medida al modelo neuronal (en 3 %), empeorando levemente respecto a *random forest*.

Las prubas que consideran solo un ensamble de modelos *random forest* ofrecen un desempeño similar al obtenido por solo uno de estos clasificadores. Esto se debe probablemente a la naturaleza de este tipo de algoritmos que es en sí mismo un ensamble.

Por lo tanto se concluye que de las pruebas realizadas hasta el momento el mejor desempeño, el cual alcanza los mayores valores en las tres métricas en una forma relativamente balanceada consiste en un ensamble de 3 clasificadores *random forest* definidos por:

- *Random Forest* con balance de clases + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

y un perceptrón multicapa estructurado de la siguiente manera:

- Red *MLP* (con tres capas ocultas de dimensión 2000, 500 y 100) + `CountVectorizer` + `CharsCountTransformer` + `EmojiHandler` + Lexicon (con parámetro `preserve_case = False`)

No obstante, se considera que puede existir aún un margen de mejora, principalmente debido a que la *accuracy* es la métrica que menos ha variado a lo largo de los experimentos y, en términos generales, se suele considerar como la más representativa e importante en cuanto al desempeño de un modelo de clasificación.

Para esto, en la siguiente sección se revisa el desempeño de métodos de *embedding* avanzados como lo son el modelo ELMo y BERT.

5.9. Experimentos con Embeddings

La prueba consiste en comparar el desempeño de los embeddings de oraciones obtenidos mediante los métodos ELMo y BERT, utilizando los mejores clasificadores obtenidos en los experimentos anteriores (*random forest*, MLP), además de probar una clasificación consistente en un ensamblaje $n = 10$ clasificadores correspondientes a sólo MLPs, sólo RFs y mixto mediante *soft voting*.

Se observa que el desempeño de BERT (AUC: 0.79; *kappa*: 0.4; *accuracy*: 0.67) resulta considerablemente superior (en cada una de las métricas de evaluación utilizadas) al obtenido mediante ELMo (AUC: 0.7; *lappa*: 0.158; *accuracy*: 0.615), siendo más consistente al realizar varias pruebas. La superioridad de desempeño del modelo BERT frente al ELMo se atribuye a que la representación de dependencia mediante mecanismos de atención resulta superior (y más eficiente) que la obtenida mediante recurrencia, además de que el modelo BERT es capaz de obtener representaciones de lenguaje más ricas al estar entrenado sobre distintas tareas al mismo tiempo (*masked language model*, *next sentence prediction*), mientras que el modelo ELMo solo incorpora representaciones contextuales bajo una única tarea. La diferencia de desempeño también puede ser fuertemente atribuida a que se utilizó el enfoque de *fine-tuning* en el modelo BERT, es decir, se re-entrenó el modelo pre-entrenado sobre un subconjunto de datos del problema a resolver, mientras que el modelo en ELMo se decidió utilizar el enfoque de extracción de características puro, es decir, el modelo no fue re-entrenado, lo cual impactaría en cierto grado en el desempeño de los algoritmos en la tarea a resolver, sin embargo, los autores enfatizan que el modelo ELMo es perfectamente funcional para extraer representaciones contextuales profundas sin la necesidad de *fine-tuning*.

Como ambos modelos buscan capturar una representación contextual profunda bi-dirieccional, se escogió la representación obtenida mediante el modelo BERT para las posteriores pruebas, con le objetivo de evitar redundancias y un aumento de dimensionalidad excesivo en el problema de clasificación.

Es importante destacar que el desempeño promedio de clasificación utilizando el *embedding* del modelo ELMo es equiparabale a la mejor combinación de clasificadores/características obtenidas en los experimentos anteriores, mientras que el desempeño promedio de clasificación utilizando el *embedding* modelo BERT resulta incluso superior, quedando de manifiesto la eficacia del enfoque de *deep learning* en comparación al enfoque de *machine learning* clásico para extracción de características relevantes en procesamiento de lenguaje natural.

5.10. Combinación de Embeddings con Representaciones Clásicas

Dados los buenos desempeños de los modelos revisados en la sección 5.9, particularmente el modelo BERT, es prometedor incorporar este al vector de características obtenido en la sección 5.8. Cabe destacar que en este contexto la función de este modelo es equivalente a la del extractor `CountVectorizer`, pues ambos consisten en una representación de la estructura del lenguaje, siendo esta última una aproximación *naive*, mientras que la primera utiliza modelos profundos para establecer las relaciones semánticas y formar así un vector de *embedding* más compacto y complejo. Por esta razón, en los experimnetos de esta sección se incurre en el reemplazo de `CountVectorizer` en el *pipeline* por el modelo profundo BERT.

Se combinó el *embedding* de representación contextual de oraciones de mayor desempeño

(BERT) con las características extraídas manualmente mediante ingeniería de características descritas en las secciones anteriores, probando los algoritmos de clasificación de mayor desempeño y distintas versiones de ensamblaje de los mismos mediante la técnica de *soft voting*.

Para los experimentos de esta sección se probaron diversas combinaciones basadas en los modelos estudiados en la sección 5.8 y extensiones de estos, aumentando la cantidad de clasificadores del ensamble así como pruebas con nuevas arquitecturas de la red *MLP*. Sin embargo, pese a que efectivamente se logran desempeños considerablemente superiores a los obtenidos en la sección 5.8 y las anteriores, ninguno de estos modelos que incorporan los *features* basados en contadores y el lexicon de Hu & Liu logró superar a lo visto en la sección 5.9. Por ejemplo, uno de los mejores y más balanceados resultados obtenidos se consiguió al considerar un ensamble de 10 clasificadores *random forest*, obteniendo una AUC del 75 %, 36 % en Kappa y una *accuracy* del 66 %. Si bien no deja de ser una mejora importante respecto a lo visto en 5.8, estos modelos son superados en forma rotunda por el modelo de la sección 5.9.

Se esperaba que la combinación de las representaciones condujera a un desempeño superior al presentado en las experiencias anteriores, ya que se incorporaría tanto la representación contextual del modelo neuronal profundo con las características intuitivamente relevantes seleccionadas manualmente. Sin embargo, los resultados mostraron que en cada experimento desarrollado, los resultados de clasificación de la combinación de representaciones resultaba levemente inferior al obtenido mediante el *embedding* del modelo BERT. Se probó normalizando las características mediante escalamiento estándar, no observando diferencias considerables a los resultados ya presentados.

La disminución en el desempeño de clasificación al combinar todas las características se le atribuye a una inherente discrepancia entre la representación obtenida mediante le modelo BERT y las extraídas manualmente, lo cual tiende a confundir a los algoritmos de clasificación, perjudicando de esta manera, el desempeño obtenido.

5.11. Selección Final

De acuerdo a los experimentos anteriores, el método de mejor desempeño resultó ser utilizando la representación de características relevantes mediante el *embedding* del modelo BERT re-entrenado sobre el dominio del problema, junto a un clasificador compuesto de un ensamblaje de 10 *random forest* mediante la técnica de *soft voting*. El modelo final fue entrenado utilizando el conjunto completo de entrenamiento provisto, alcanzando el primer lugar en cada métrica de evaluación de la competencia de clasificación de intensidad de sentimientos en tweets, obteniendo un desempeño promedio de 0.79 en *AUC*, 0.442 en coeficiente *kappa* y 0.684 en *accuracy*.

Si bien pudiera considerarse que el desempeño obtenido por este último modelo es aún mejorable, teniendo en cuenta lo sofisticado y especializado del modelo empleado vale la pena volver al propio *dataset*. Como se dijo con anterioridad, se realizó una inspección visual de los datos disponibles para entrenamiento para las cuatro emociones a categorizar. Más allá de errores pequeños como datos duplicados y la presencia de algunos ejemplos que podrían considerarse como *outliers*, cabe tener en mente que pese a las medidas tomadas los conjuntos presentan un desbalance y son de reducido tamaño, si se considera la cantidad de documentos que típicamente se recoge para tareas en *Natural Language Processing*. Tener una escasa cantidad de documentos puede reper-

cutir en la cantidad de información disponible y la capacidad de generalización de los modelos y, en el peor de los casos, ser una muestra no representativa del fenómeno estudiado (incluso que el mismo problema esté mal planteado). En este sentido se percibieron varios casos patológicos, particularmente en los datos asociados al sentimiento *fear*. Estos ejemplos contenían el término *awe* (temor en inglés), pero al analizar dichas oraciones es extremadamente poco probable que se haya utilizado la palabra *awe* bajo esa acepción, puesto que se perdía el sentido gramatical de la oración, y el tema de la misma era de una índole bastante diferente al sentimiento de temor.

6. Conclusiones

A lo largo del presente trabajo se presentó la problemática de clasificación de intensidad de emociones, particularmente intensidades *alta*, *media* y *baja* para las sensaciones de *ira*, *miedo*, *alegría* y *tristeza*. Para casa sentimiento y en forma separa unos de otros, se plantea el problema como uno de clasificación en tres clases. Tras formalizar las principales métricas en base a las cuales se medirá el desempeño de los modelos estudiados. Tras esto se revisan en forma breve los algoritmos tanto de clasificación como de extracción de características empleados a lo largo de este proyecto.

Se procede a realizar experimentos, particularmente bajo dos enfoques: el primero estudiar el diseño de algunas representaciones y probar si efectivamente conllevan alguna ganancia en *performance*, para luego combinarlas en forma incremental tanto entre ellas como con algoritmos clásicos de aprendizaje de máquinas, considerando técnicas de ensambles. Esto último fue un factor determinante para lograr un balance entre las 3 métricas de desempeño a la vez que se incrementaban las mismas. En este aspecto, la evaluación y comparación de métricas resultó ser un factor esencial, permitiendo interpretar en una forma más completa el desempeño tanto de los *features* considerados como los propios algoritmos utilizados. Dado que cada métrica se basa por sí sola en principios distintos, permite reflejar comportamientos más diferenciados respecto al comportamiento de los modelos probados. De esta forma se pueden fomentar comportamientos deseables pero diferentes en forma simultánea, o evidenciar que esto de hecho no ocurre, lo cual puede fundamentar un cambio de modelo en busca de uno mejor.

Una vez que se escogió el modelo "óptimo" bajo este primer enfoque, basado precisamente en las representaciones más informativas encontradas y con clasificadores *random forest* y perceptrón multicapa (los cuales corresponden de hecho a los con mejor expresividad de entre los algoritmos clásicos estudiados), se procede a tratar el segundo enfoque, basado en técnicas de *Deep Learning* modernas y especializadas para tareas en *Natural Languaje Processing*, con la finalidad de incorporar al modelo una representación más densa y que pueda capturar en mejor medida las relaciones semánticas presentes en los datos. Esto significó una mejora aún más pronunciada en el desempeño de los clasificadores. Contrario a lo que cabría esperar, la combinación de esta representación densa con los *features* obtenidos en base a los extractores de caracterísiticas diseñados significó una pérdida de desempeño. Se atribuye esto a la discrepancia generada por dicha combinación, mezclando contadores y asignando connotaciones a términos individuales con representaciones que capturan la información contextual y semántica contenida en los mensajes.

Durante el desarrollo de la experiencia se tuvo que lidiar con una gran cantidad de dificultades, tales como la presencia de un conjunto de datos acotado y desbalanceado, con casos bordes de tweets mal o confusamente etiquetados, lo cual sumado a la complejidad y diversidad inherente del lenguaje natural, resultaron en una tarea de clasificación de altísima dificultad, lo cual explica en parte el bajo desempeño de los métodos de *machine learning* clásicos enfocados en selección de características, donde la incorporación de *features* considerados fundamentales no reflejaron un impacto significativo en los resultados obtenidos. Ejemplo de esto es el uso de mayúsculas y elongación de palabras, los cuales podrían ser considerados como relevantes en tareas similares a la abordada, cosa que no es reflejada por el conjunto de datos de entrenamiento y, por tanto se vuelven irrelevantes en términos del aprendizaje. Sin embargo, se apreció que los algoritmos

basados en *deep learning*, capaces de seleccionar características relevantes de patrones presentes en los propios datos, resultaron mucho más propicios para resolver la tarea designada, siendo capaces de interpretar sesgos no evidentes del conjunto de datos utilizado (tales como el etiquetado poco intuitivo en ejemplos bordes), obteniendo un nivel de desempeño muy superior en comparación al enfoque anteriormente mencionado, el cual, sumado a un proceso de clasificación basado en el ensamblaje de múltiples algoritmos, resultaron en una predicción de certera y robusta a sobreajuste.

Como trabajo futuro para mejorar el desempeño de clasificación de la tarea descrita se propone investigar e incorporar algoritmos del estado del arte actual para la obtención de características relevantes, tales como el modelo XLNet [10], explorar la utilización directa de distintos modelos neuronales profundos re-entrenados mediante *fine-tuning* en la clasificación en un amplio ensamble (de alto costo computacional), una búsqueda de hiperparámetros de los modelos de extracción de *embeddings* y clasificación de mayor rigurosidad, como por ejemplo una búsqueda Bayesiana, además de realizar un pre-procesamiento más prolijo del texto para facilitar la extracción de características relevantes, enriqueciendo la traducción de acrónimos o implementando un método efectivo de separación de *hashtags* compuestos (distintos conceptos sin separación explícita), entre otros.

7. Referencias

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[3] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] A. Fatir. (2019) Sentiment analysis on tweets. [Online]. Available: https://github.com/abdulfatir/twitter-sentiment-analysis/blob/master/code/preprocess.py

[9] M. Hu and B. Liu, "Mining and summarizing customer reviews," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[10] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[12] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. Peters, M. Schmitz, and L. S. Zettlemoyer, "Allennlp: A deep semantic natural language processing platform," 2017.

[13] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.