



UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA

INFORME DE TRABAJO PROFESIONAL

MAGUS  
REAL TIME SENTIMENT FORECAST BASED ON  
TWITTER STREAM

Martinez, Gaston Alberto – 91383

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Descripción del problema</b>	<b>2</b>
<b>3. Objetivos</b>	<b>4</b>
<b>4. Características del trabajo</b>	<b>4</b>
4.1. Análisis previo del problema . . . . .	4
<b>5. Desarrollo del Trabajo</b>	<b>7</b>
<b>6. Definición del modelo del clasificador</b>	<b>7</b>
6.1. Attardi (Estructura base) . . . . .	8
6.2. Attardi (Estructura modificada) . . . . .	9
6.3. Estructura propuesta . . . . .	9
6.4. Entrenamiento . . . . .	13
6.5. Configuraciones de hyperparametros . . . . .	14
6.6. Métricas . . . . .	14
6.6.1. F1 Score . . . . .	15
<b>7. Features</b>	<b>16</b>
7.1. Features habilitadas . . . . .	16
7.1.1. Palabras . . . . .	16
7.1.2. Caracteres . . . . .	17
7.2. Features descartadas . . . . .	17
7.3. Preprocesamiento . . . . .	17
7.3.1. Anonimizacion del tweet . . . . .	18
7.3.2. Reducción de repeticiones . . . . .	18
7.3.3. Eliminación de tildes . . . . .	18
7.3.4. Eliminación de las mayúsculas . . . . .	18
7.3.5. Motivación . . . . .	19
7.4. Normalizacion de features mediante Embedings . . . . .	19
<b>8. Conformacion del corpus</b>	<b>21</b>
8.1. Crowdsourcing . . . . .	21
8.1.1. Definición de la validez de una clasificación . . . . .	22
8.2. Corpus utilizados como base . . . . .	22
8.2.1. Corpus en inglés . . . . .	22
8.2.2. Corpus en español . . . . .	22

<b>9. Pruebas preliminares del modelo</b>	<b>23</b>
<b>10. Implementacion</b>	<b>25</b>
10.1. Magus (Pipeline) . . . . .	25
10.1.1. Kuhn (Master) . . . . .	26
10.1.2. Magus Core (Worker) . . . . .	26
10.1.3. Modelo distribuido . . . . .	27
10.2. Morgana (Clasificador) . . . . .	27
10.3. Alkaid (Web Tagger) . . . . .	28
10.4. The World (Mapa en tiempo real) . . . . .	30
10.5. Yata (Trainer) . . . . .	31
<b>11. Evaluación del modelo</b>	<b>32</b>
11.1. Primera evaluación . . . . .	32
11.1.1. Accuracy . . . . .	33
11.1.2. F1-Scores . . . . .	37
11.2. Evaluación final . . . . .	40
11.2.1. Accuracy . . . . .	41
11.2.2. F1-Scores . . . . .	44
<b>12. Conclusiones finales</b>	<b>45</b>
<b>13. Trabajo a futuro</b>	<b>46</b>

## 1. Introducción

El siguiente trabajo se presenta en el marco del desarrollo del Trabajo Profesional para la carrera Ingeniería en Informática del estudiante Gaston Alberto Martinez. Con el objetivo de aplicar los conocimientos adquiridos durante la carrera, se eligió desarrollar el temas “**Análisis de sentimientos más allá de la polaridad sobre el *stream* de Twitter**”.

## 2. Descripción del problema

El análisis de sentimientos es un proceso que se utiliza para determinar si un texto contiene una opinión del escritor. Usualmente, se traduce en una clasificación positiva o negativa del mismo y suele emplearse en reseñas o redes sociales. El análisis aplicado a estas últimas resulta de gran interés, a tal punto de ser motivo de competencias.<sup>1</sup> En la práctica estos análisis abarcan desde el conocimiento de la opinión de un conjunto de personas sobre un tópico en particular hasta determinar problemas con algún servicio de forma completamente automática. El creciente volumen de datos y aplicaciones hacen a este tipo de análisis muy interesante y valioso.

Twitter es uno de los servicios de *microblogging*<sup>2</sup> más conocidos y utilizados en la actualidad. En él los usuarios plasman situaciones de sus vidas diarias, comunican emociones e ideas y transmiten espontáneamente cualquier pensamiento que les surja en el momento.

Increíble Egipto, impresionante, locura, se cae el estadio, están en Rusia 2018 con gol al minuto 95, se los habían empatado. Te amo fútbol.

-@RiverettiMX (8 oct. 2017)

De esta manera, Twitter resulta una fuente de conocimiento acerca del estado emocional de las personas, especialmente en lo que respecta a los acontecimientos de carácter masivo. Es sencillo inferir la opinión de un conjunto de la población sobre un tópico en particular (o la manera en la que los afecta) solo leyendo los tweets relacionados con el mismo.

---

<sup>1</sup><http://alt.qcri.org/semeval2017/task4/>

<sup>2</sup>Sitio web que incluye, a modo de diario personal de su autor o autores, contenidos de su interés, actualizados con frecuencia y a menudo comentados por los lectores. [1] Los sistemas de *microblogging* se caracterizan por limitar el número de caracteres de cada entrada, usualmente a no más de 140.

No obstante, Twitter ya no se utiliza únicamente como red social de *blogging*. Se le da uso también como herramienta de publicidad, de noticias o simplemente de compartición de fotos.

Mendoza: La reacción de un grupo de vecinos con un auto estacionado sobre la senda peatonal. <https://goo.gl/GuEDrg>

-@C5N (8 oct. 2017)

Este tipo de tweets son los que se denominan *tweets neutros*, los cuales no permiten inferir información sobre el emisor o al menos, no solamente en base al texto.

El trabajo constará, en primera instancia, en separar los tweets neutros de los restantes. Luego en clasificar los no filtrados de forma automática entre las diferentes clases definidas. Además, como este tipo de análisis tiene un valor mayor si se lo realiza en tiempo real, es necesario transformarlo en un requerimiento necesario para darle un valor agregado al trabajo.

Las formas más comunes de lograr una buena performance en el procesamiento de datos infinitos suelen ser a través de la utilización de arquitecturas del tipo *pipeline*, como las que se pueden encontrar en los microprocesadores. Estas se caracterizan por separar el proceso en etapas que funcionan de forma independiente, siendo solo necesario resultado de la etapa anterior. Este tipo de construcciones es naturalmente paralelizable y permite procesar una mayor cantidad de datos en simultáneo, impidiendo que las partes más lentas del proceso ralenticen a todo el procesamiento global. Las mismas ya se han utilizado con éxito en el campo del procesamiento del lenguaje natural como, por ejemplo la denominada TANL Pipeline de Giuseppe Atari.[2]

Dichas arquitecturas, al ser partes independientes que únicamente consumen datos y producen nuevos, son fácilmente distribuibles en distintos equipos. Su capacidad de escalar horizontalmente, utilizando los mecanismos indicados, permite mejorar significativamente el volumen de datos procesados en simultáneo.

### **3. Objetivos**

El trabajo se desarrolla con vistas a cumplir los siguientes objetivos:

- Desarrollar un sistema capaz de clasificar textos obtenidos a través de un stream infinito, de forma eficiente.
- Diseñar y analizar una alternativa viable a los clasificadores polares.
- Mostrar los resultados de forma intuitiva y en tiempo real.

### **4. Características del trabajo**

#### **4.1. Análisis previo del problema**

El punto de partida es la propuesta de tesis de Phillip Smith,[3] en la cual se hace un análisis de sentimientos multivalorado. El enfoque propuesto por la misma es utilizar clustering para la clasificación y difiere en nuestro objetivo en varios puntos clave:

- Es un solo lenguaje el que se analiza.
- Es un clasificador estático, lo cual implica que no importa cuándo estén los resultados sino que estén.
- Asume buena redacción/sintaxis, de lo cual no existe garantía en Twitter.
- Se ciñe únicamente a notas de suicidio (tópico uniforme).

En nuestro caso, el ambiente multilenguaje de por sí supone un nivel de datos muy grande como para poder manejarlos efectivamente mediante *clustering*; sin contar el hecho de que el volumen de features es exponencialmente grande y que vuelve la tarea mucho más engorrosa. Por estas razones se descarta ese enfoque a priori, pero se continúa considerando el análisis hecho en el paper ya que provee un esquema útil para la conformación de las clases de sentimientos.

Existen al menos dos cuestiones claves: el qué y el cómo. No siempre resulta tan importante lo que se dice sino también cómo se lo dice. Esto último ayuda a distinguir ciertas subjetividades en el discurso. Por ejemplo, se puede enfatizar un estado de emoción violenta a través de las mayúsculas mientras que, la misma frase en minúsculas, atenúa la intensidad de ese sentimiento puntual para el lector. Además de las dos cuestiones mencionadas,

existen otros mecanismos que le dan connotaciones subjetivas a la misma construcción sintáctica. El con qué es parte de ello: el uso de *emojis* puede ser indicador de ironía en una oración ó restarle importancia a su significado literal. “Hoy me robaron” y “hoy me robaron =P” son drásticamente distintas. Este tipo de construcciones son dependientes del idioma, del entorno sociocultural del que escribe y del público al que quiere dirigirse. En un ambiente multilenguaje resulta imposible realizar a mano dicha clasificación ya que, en un entorno donde el espectro de emociones es más amplio y específico, apuntar simplemente a la sintaxis es un abordaje insuficiente, aunque, por otra parte sirve para asociar fácilmente un discurso a una escala de solo positivo-negativo. Entonces, cuando se intenta abarcar un mayor espectro de matices, resulta más difícil lograr un buen resultado sin agregar otros indicadores. Un claro ejemplo se da en el empleo de una lengua extranjera: un estadounidense diciendo “taco taco taco” posee una connotación distinta a la que podría guardar si lo hiciera un mexicano (es comúnmente utilizado como burla cuando alguien habla en español en un ambiente donde predomina el inglés).

De esta manera, se puede concluir que un enfoque clásico basado en lexícones no resulta un buen approach: por un lado, porque realizarlo a mano es físicamente imposible y, por el otro, porque si se emplearan técnicas de construcción automática de lexícones se dejaría de lado todo el significado implícito de cada palabra. En línea con lo anterior, un enfoque orientado a información contextual como los basados en *Word2vec* y en redes neuronales, se ve mucho más razonable.

Sin tener en cuenta el *Word2vec*, los enfoques de redes neuronales tienen un cuello de botella muy importante: la entrada de datos. Si uno no puede garantizar que un *feature map* sea acotado, no se puede crear y entrenar una red con ello. Por lo tanto si se acepta todo como un feature, los feature maps son infinitos. Incluso con el tiempo aparecen dinámicamente nuevas features a medida que aumenta el número de muestras disponibles. Claramente se hace imposible elaborar un sistema escalable acorde a las necesidades usando este enfoque. A primera vista el hash trick sería una solución pero la realidad es que tiene al menos un problema, y no menor: a medida que se agranda el universo de elementos que se pueden analizar (es un stream) llega indefectiblemente el momento en el que surge alguno que no puede ser clasificado. Utilizando el hash trick, al perder información de qué feature es en el feature map, se pierde la capacidad de conocer eso de antemano y, incluso puede matchear con algo que no corresponde. Descartando el hash trick se termina en una situación en la cual no se puede avanzar, dado el infinito número de features. Entonces, alternativas a eso utilizarían *clustering* para matchear cada feature a un índice distinto de la entrada de la red, indicando la presencia

de un grupo de features o no. Si bien esto parece una buena opción, el clustering clásico es lento y el agrupar según frecuencias de aparición en cada una de las clases supone un vía importante en la entrada de la red. Por lo tanto, Word2vec surge como una opción más útil para reducir las dimensiones a la entrada de la red. Incluso no soluciona a priori el problema de las *features infinitas*, es decir, lo acota pero no lo resuelve por sí mismo.

Ahora bien, con una pequeña modificación en la forma de trabajo se puede utilizar este mecanismo efectivamente. Word2vec cumple una doble función: por un lado da una representación acotada de cada feature, con lo que el tamaño de la entrada se puede fijar en una de sus dos dimensiones (ancho). Por el otro, codifica dentro de esa representación la información contextual del feature. Estas dos características son ampliamente buscadas para el enfoque a adoptar. Aun así, por si solo sigue sin ser la solución alternativa al problema de las features infinitas. Recordando, se busca usar features que vayan más allá del simple significado de la palabra, pero aun se continúa sin poder acotar el número total de features a la entrada. Añade un nuevo problema: si se planteara utilizar el mismo Word2vec para codificar palabras y otros elementos por un lado el algoritmo tardaría demasiado y, por el otro, encontraría relaciones inútiles que podrían afectar el resultado final. Por ejemplo, siempre que se encuentre “HOLA”, se va a encontrar “hola” (suponiendo 2 tokenizers, uno sin pre-procesamiento y otro que translada todo a minúsculas). Ahora, es claro que “hola” todo en mayúsculas no se encuentra necesariamente relacionado con el significado de “hola”.

Asimismo, se puede pensar lo siguiente: en un tweet no puede haber más de 70 palabras. Las palabras en sí codifican el sentido semántico del tweet; mientras que las palabras mas el idioma de origen del tweet, las palabras más si son hashtags y demás, codifican la información meta sintáctica del tweet. La cantidad de cada tipo de feature está acotada completamente debido a la existencia de una longitud máxima en los tweets. Entonces, se puede separar el embedding de cada grupo de features según su rol, con lo que, de esta forma, se achican los espacios y las features guardan real relación entre sí. Así también, se puede determinar la cantidad máxima de features de cada tipo que pueden aparecer, con lo que finalmente se simplifica el feature map.

Es más, incluso se pueden agregar otras features de soporte, así como embeddings a nivel carácter. A pesar de que se demostró que por sí mismos los embeddings a nivel carácter no producían buenos resultados, se los puede mantener como información de respaldo. En un contexto donde la sintaxis está lejos de ser perfecta, se puede utilizar cierta información de los caracteres utilizados para aproximar a la clasificación esperada.

A través de esta metodología, se puede utilizar un conjunto arbitrario de tokenizers y tener siempre un feature map de tamaño fijo y un embedding

representativo de cada feature generada. Para poder explotar Word2vec de esta forma, es necesario poder representar cada feature como una cadena de forma inequívoca dentro de su espacio de features. El resto es simplemente una variación de ajustes de los metaparametros del Word2vec, como, por ejemplo, el tamaño de la ventana.

## 5. Desarrollo del Trabajo

Dado el análisis previo y estado del arte,[4] se utilizo un enfoque basado en redes neuronales. El mismo se fundamenta en la siguiente afirmación:

*Como personas a nosotros nos cuesta per se identificar la categoría de cada tweet, pero más aún, cuesta los factores que hacen a nuestra elección.*

Las redes neuronales en sí, son buenas para identificar patrones, sin necesidad de una intervención humana directa. Además, estos patrones pueden ser completamente imperceptibles para un humano, en el sentido de que hay cosas que tenemos tan naturalizadas que no notamos como nos afectan subconscientemente durante la toma de decisiones. Un ejemplo de esto podría ser el uso de determinadas palabras, que reemplazadas por un sinónimo léxico gráficamente válido, cambian el sentido de la oración.

## 6. Definición del modelo del clasificador

El modelo propuesto surge de utilizar la misma estructura de red que red que utiliza Attardi,[5] para la clasificación polar. Obviamente, se le deberán realizar algunas modificaciones de forma tal que acepte los diferentes outputs.

## 6.1. Attardi (Estructura base)

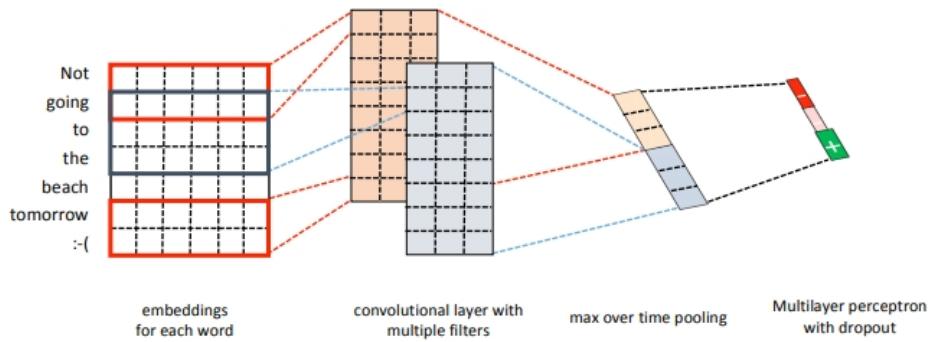


Figura 1: Arquitectura propuesta en Convolutional Neural Networks for Sentiment Classification

La Estructura base propuesta por Attardi, se puede observar en la figura 1, y consiste de:

- Capa de entrada: Transforma las cadenas recibidas en su representación matricial correspondiente mediante el calculo de sus embeddings a nivel palabra. Los embeddings se calculan en un sub-espacio de dimensión 300.
- Capa de filtros: Dada la matriz que representa al texto, aplica  $m$  juegos de  $n$  filtros, cada uno de tamaño  $h_i$  a la misma para obtener las features. Cada uno de estos features que consisten de una matriz columna, son luego pasados por una capa de max pooling que conforma finalmente el feature extraido. En el paper se propone utilizar 4 juegos distintos de tamaños 3, 5, 7, 7 cada uno con 128 filtros.
- Capa de dropout: Capa de filtrado con dropout rate del 0.5
- Capa densa full connected: Dadas las features extraídas y concatenadas, genera una matriz columna con la asociación del texto a cada uno de los grupos.

La capa de filtros es activada vía relu y la capa densa no tiene función de activación.

## 6.2. Attardi (Estructura modificada)

Para que el análisis sea efectivamente válido, se propuso modificar los hiperparámetros de la red base a modo de buscar la mejor performance que esta puede dar para el nuevo caso de uso.

Por empezar, los embeddings ya no se calculan dentro de la misma red, con lo cual la entrada ahora es la matriz que representa al texto. Se modifico también de forma acorde el tamaño de los embedigns ya que es necesario que codifiquen un mayor número de cosas.

Se modifico también la cantidad de juego de filtros, sus respectivos tamaños y cantidad de filtros por juego.

## 6.3. Estructura propuesta

La clave detrás del enfoque propuesto radica en que dependiendo del tipo de feature que se utilice para realizar el análisis, se pueden captar distintos aspectos del mismo tweet. Cada una de estas representaciones”, permiten detectar cosas que otras no, y por lo tanto, la combinación de todas ellas permitiría poder hacer un análisis mas preciso.

Teniendo esto en cuenta, se utiliza como base la estructura antes expuesta para generar una estructura más compleja y abarcativa. Para cada tipo distinto de feature, se monta un “stream”, que consiste en una estructura similar a la base: una capa de convolución, una capa fully connected grande y una capa fully connected de salida (tamaño de las clases). Todos los streams se conectan luego a otros dos streams que hacen la clasificación final, uno conectado a las capas intermedias (global) y otro conectado a las capas finales.

Estructuras con una lógica similar ya se utilizaron en otros trabajos, en particular *A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks*[6] sirvió de inspiración para este trabajo. La diferencia entre la implementación del paper y la propuesta radica en que en el primer caso, se utilizan redes pre-entrenadas con outputs de distinta clase a los esperados a obtener en el clasificador; mientras que en este caso, las redes intermedias no están pre-entrenadas y sus outputs son consistentes con los del clasificador. Este último punto es el motivo por el cual se optó por introducir una capa intermedia antes de la capa de salida.

El esquema de la estructura se puede ver en la figura 2 a nivel global y en las figuras 3 4 5, se puede apreciar el esquema interno de cada stream.

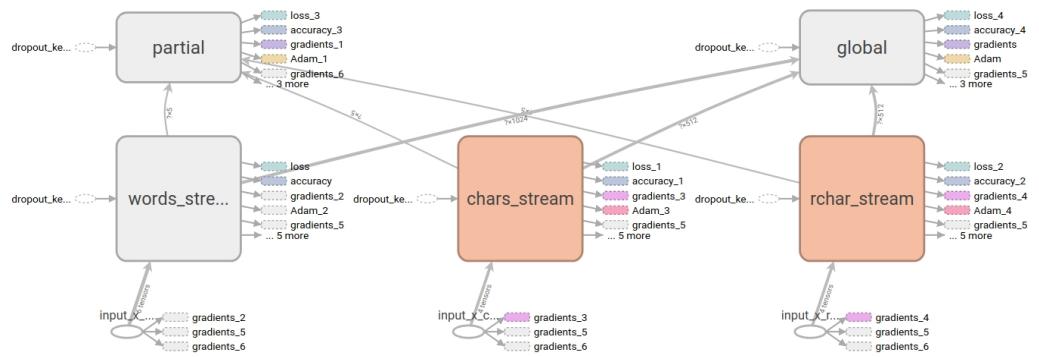


Figura 2: Esquema de red propuesto

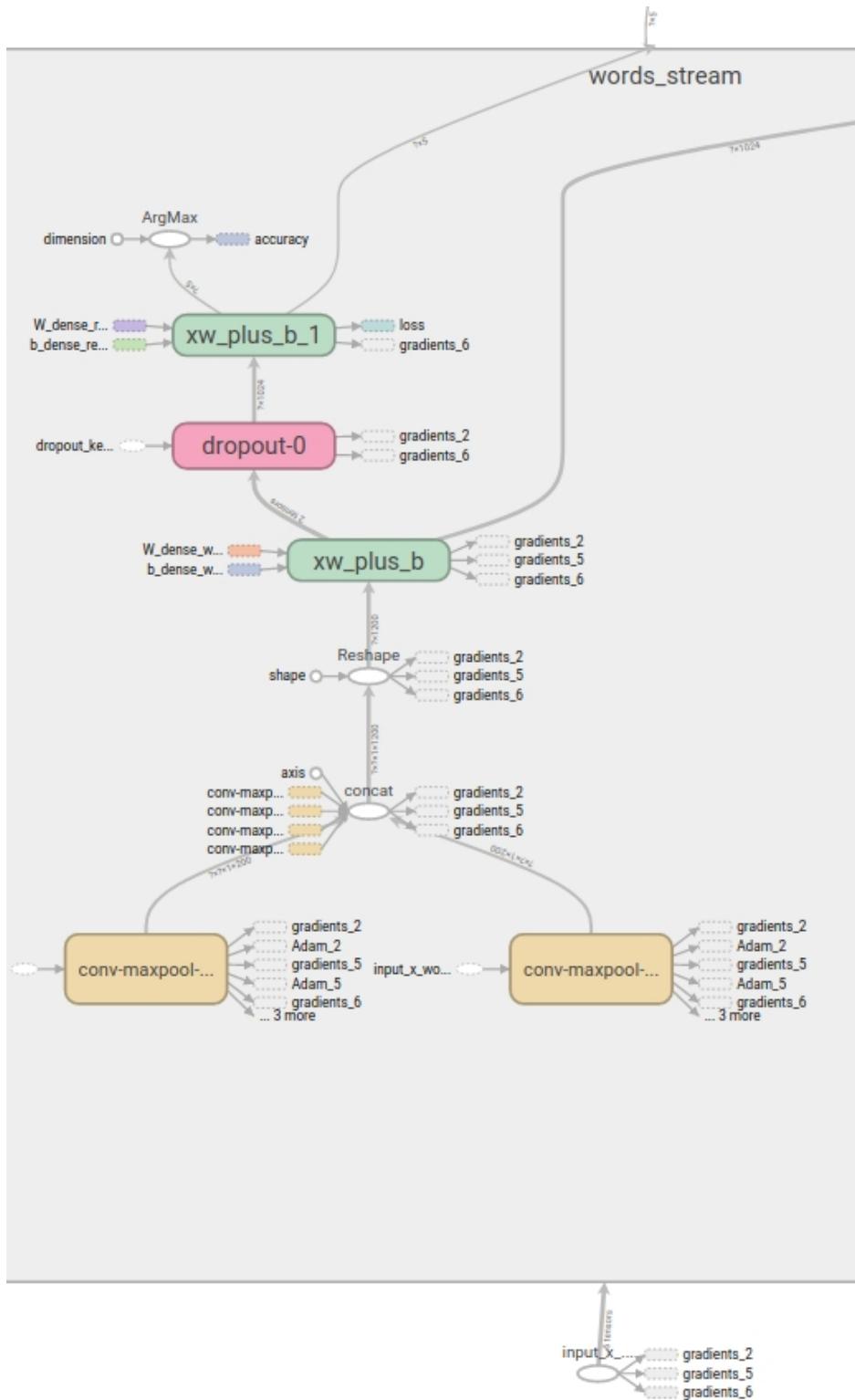


Figura 3: Esquema interno de los streams intermedios

Todos los streams internos tienen la misma estructura, una capa de convolucion con max polling, conectada a una capa densa oculta, finalmente conectada a la capa densa de salida mediante un dropout. El stream de salida Global (figura 4) se conecta a las capas ocultas mientras que el stream de salida Partial (figura 5) se conecta a las capas de salida.

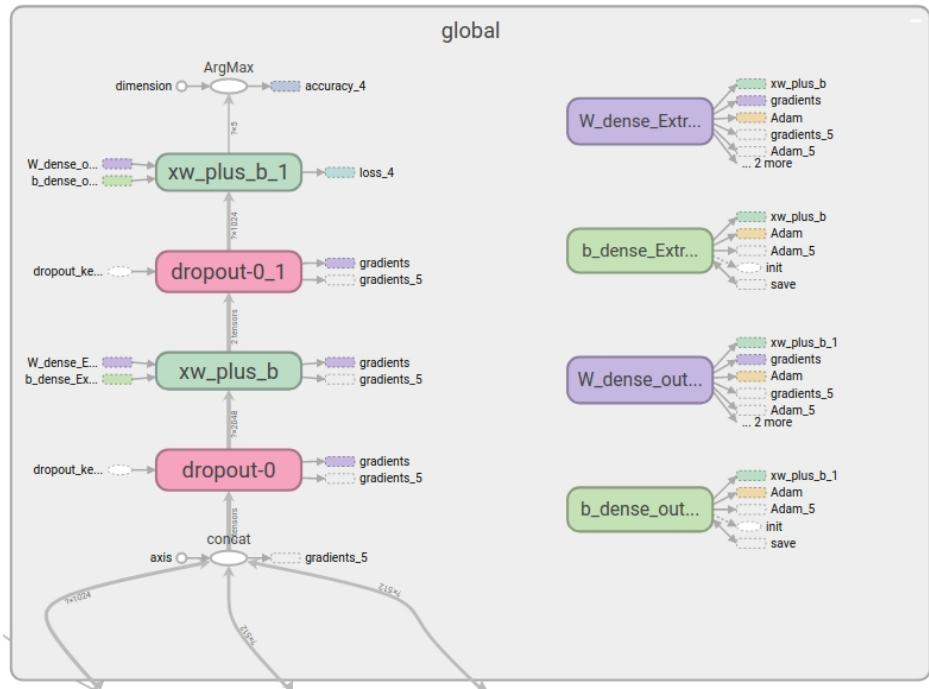


Figura 4: Esquema interno del stream Global

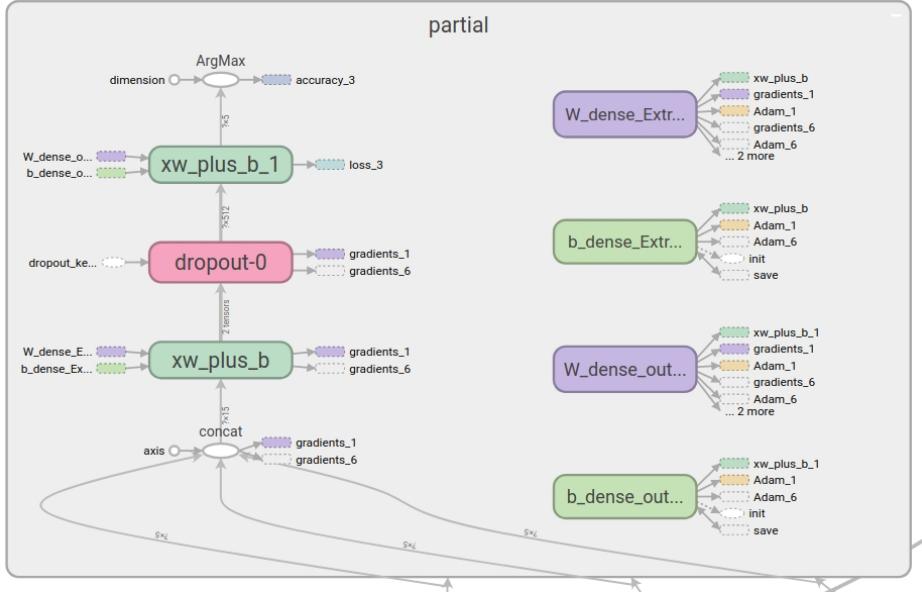


Figura 5: Esquema interno del stream Partial

La diferencia entre ambos radica únicamente en que el stream Global al recibir datos de una capa intermedia, tiene una capa de dropout a la entrada del mismo.

#### 6.4. Entrenamiento

El objetivo del diseño es poder entrenar cada stream de forma independiente, buscando que aprenda a clasificar por su cuenta. De esta forma, el entrenamiento no solo es mas rápido al tener que actualizar menos variables, sino que evita que los streams se vuelvan co-dependientes, es decir, que aprendan solo para mitigar las clasificaciones de los otros.

Para lograr este efecto, las iteraciones del entrenamiento se organizan de la siguiente forma:

- Cada iteracion es un pasaje de un batch atravez de los tres streams internos.
- Si durante esta iteración, alguno de los streams llega a tener una precision promedio mayor al 60 %, se entrena los streams de salida durante un epoch. Esto se realiza de esta forma, porque no tiene sentido entrenar los streams de salida sobre algo aun no clasifique bien.

- Si se cumplieron un número específico de iteraciones, se realiza un paso de prueba, en el cual se mide la precisión del clasificador sobre el set de test.

## 6.5. Configuraciones de hiperparámetros

Para la configuración de los hiperparámetros, primero se utilizaron las configuraciones base propuestas en la estructura base. En base a esa configuración, se realizaron las modificaciones pertinentes, primero buscando una configuración en la cual la red lograra aprender, y luego tratando de reducirla para que logre efectivamente generalizar.

Se optó por seguir las recomendaciones y metodologías sugeridas. [7] Según estas, es necesario solo detenerse a evaluar los parámetros de tamaño de filtro y cantidad de filtros ya que son los que mayor influencia sobre el resultado tienen. Como mecanismo de pooling, según los análisis hechos en el citado paper, usar algo que no sea 1-max pooling afecta negativamente. El mecanismo de regularización utilizado tampoco juega un papel significativamente importante en el resultado final, y solo llega a tener relevancia para ciertas configuraciones de los otros parámetros. Como función de activación se utilizara tanh ya que es la que mejores resultados muestra en el citado paper, y no hay notables diferencias contra *relu*.

## 6.6. Métricas

Un punto importante a definir al momento de entrenar la red, es el mecanismo de evaluación que se va a utilizar para determinar la eficiencia de la misma. En un principio, la métrica de precisión simple (clasificados correctamente sobre total de muestras a clasificar) se veía como una alternativa razonable. La realidad, es que en algunos casos, si bien la métrica daba bien para el set de pruebas, al utilizarlo en un entorno real, tenía un mucho peor desempeño.

La razón de esto es que el corpus no es balanceado, es decir, que las muestras disponibles para cada clase difieren mucho en cuanto a cantidad. Con un corpus desbalanceado, la precisión simple, termina no dando una idea real de que tan bien se desempeña sobre todas las clases. Por ejemplo si el 90 % del corpus pertenece a la clase A, y el 10 % restante a las otras clases, la precisión podría ser del 90% aun cuando solo clasifique todo al conjunto A. Una alternativa para mitigar este problema seria la construcción de un corpus balanceado. No obstante como se explica en la sección 8, los datos son naturalmente desbalanceados y eso no es una tarea sencilla aun contando con la capacidad humana necesaria.

### 6.6.1. F1 Score

Como se explica en [8], para un corpus desbalanceado, una mejor métrica seria “F1 Score”, que tiene en cuenta no solo que tan bien clasifica a nivel global, sino que pondera los resultados según la cantidad de muestras totales de cada clase. Para poder calcular el “F1 Score” se debe definir precisión y recall como:

$$precision = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos} \quad (1)$$

$$recall = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos} \quad (2)$$

Luego, utilizando esto, la formula del F1 Score es:

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} \quad (3)$$

$$= 2 * \frac{precision * recall}{precision + recall} \quad (4)$$

Siguiendo el ejemplo anterior, para un clasificador que clasifica todo A, la clase tendría un F1 de aproximadamente .94 y el resto de las clases un F1 de 0, con lo cual el F1 de todo el clasificador para 5 clases seria menor a 0.2. Ahora si da una idea de que la red no clasifica correctamente, y por ende se tiene ahora si una métrica mucho mas realista.

Ahora bien, la fórmula expresada de “F1 Score” requiere poder discriminar entre verdaderos positivos, falsos positivos y falsos negativos para cada clase. La forma de obtener estos datos para poder calcular los “F1 Score” de forma eficiente es mediante la conformación de una confusión matrix.

		Actual class		
		Cat	Dog	Rabbit
Predicted class	Cat	5	2	0
	Dog	3	3	2
	Rabbit	0	1	11

Figura 6: Ejemplo de confusión matrix para 2 clases

		Actual class	
		Cat	Non-cat
Predicted class	Cat	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

Figura 7: Datos extraídos de la confusión matrix anterior para la clase Cat

En cada celda de las confusión matriz se acumulan la cantidad de casos encontrados de elementos de la clase *columna* clasificados como *fila*. Esos datos luego representan:

- Diagonales: Verdaderos positivos
- Columnas sin los elementos de la diagonal: Falsos positivos para la clase de la columna
- Filas sin los elementos de la diagonal: Falsos negativos para la clase de la columna

Con estos datos ya se puede calcular la precisión y el recall de cada clase, según las formulas (1) y (2) respectivamente, para luego poder si sacar su correspondiente “F1 Score” con (4)

## 7. Features

Del numero total de tipos de features pensadas utilizar originalmente, se utilizaron solo tres: palabras y caracteres (con y sin pre-procesamiento).

### 7.1. Features habilitadas

#### 7.1.1. Palabras

Las palabras como features están pensadas para obtener la información sintáctica del tweet. Ya que solo importan las palabras, las mismas deben ser normalizadas de alguna forma para que sean comparables entre sí. Por ejemplo, sistemáticamente, “casa” y “CASA” representan lo mismo.

### **7.1.2. Caracteres**

Los caracteres como features apuntan a dos grandes objetivos:

- Compensar los errores de tipeo o mala escritura.  
Al poder ver los caracteres únicamente, podría llegar a identificar partes relevantes de las palabras, como ocurre por ejemplo al aplicar algún método de stemming<sup>3</sup>.
- Poder captar aspectos mas subjetivos de los tweets  
Esta es la razón por la que se utiliza el feature “raw char” (caracteres sin procesar). Por ejemplo, este tipo de features podría detectar el enojo a través del uso excesivo de mayúsculas.

El uso de features de tipo carácter ya fue estudiado como mecanismo para mitigar el efecto de las features no presentes en los sets de entrenamiento [9] y como única feature [10].

## **7.2. Features descartadas**

Varias de las features pensadas para utilizar originalmente fueron descartadas para ser utilizadas en este trabajo. Esto se debe en mayor medida al hecho de que los tweets pre-clasificados disponibles, no contenía toda la información necesaria para poder construir dichas features (por ejemplo, contener país de origen de alguna forma).

En menor medida, fue porque las evaluaciones previas dieron a ver que no sería necesario agregarlas. Algunas de las features originales estaban pensadas para mitigar el efecto del multilenguaje, pero en las pruebas preliminares la red funcionó de forma similar que en el caso de solo inglés. Lo anterior sumado al hecho de que agregar más features implica necesariamente mayor tiempo de procesamiento, entrenamiento, y que esto no es deseado, derivó en la no utilización de dichas features.

## **7.3. Preprocesamiento**

Antes de poder ser realizada la extracción de features, es necesario preprocesar el contenido del mismo.

---

<sup>3</sup>Los algoritmos de stemming permiten extraer de una palabra su raíz, por ejemplo, eliminando conjugaciones o plurales.

### **7.3.1. Anonimizacion del tweet**

Se entiende por anonimizar un tweet al proceso de reemplazar las menciones y los links por tokens representativos. Las menciones se reemplazan por el token “@USER” y las url por “HTTP://URL”, de esta forma, se garantiza:

- Todas las menciones y todas las url matchearan a los mismos features
- Reducir el ruido, ya que las features que utilizan caracteres son muy susceptibles de ser afectadas por las urls
- Garantizar la privacidad de los datos

### **7.3.2. Reducción de repeticiones**

Es comun en linea utilizar la repeticiones de caracteres por diversos motivos. Si bien hay diferencias entre el uso correcto de la palabra y el uso con repeticiones (por ejemplo para enfatizar), no existe real diferencia entre por ejemplo la misma palabra con el ultimo caracter repetido diez o quince veces. Por ello, se reducen las repeticiones de caracteres consecutivos a un máximo de tres repeticiones.

### **7.3.3. Eliminación de tildes**

La escritura en Internet no se suele caracterizar por el correcto uso de la gramática y no es extraño no encontrar tildes en el común de los casos. Si bien se dice que del uso de la correcta escritura en linea indica algún tipo de animosidad hacia el receptor, es un análisis que se puede hacer solo teniendo el historial de mensajes del emisor, con lo cual, no necesariamente es una generalización valida. Por ello, se eliminan las tildes de las palabras, de modo de poder garantizar que ambas representen la misma feature.

### **7.3.4. Eliminación de las mayúsculas**

Al igual que el ítem anterior, para el significado semántico de la palabra, si la misma esta escrita con mayúsculas, minúsculas o intercalado. Entonces se decidió pasar todo el tweet a minúsculas (dejando los tokens especiales en mayúscula a modo de poder identificarlos), para así generar una uniformidad entre los diferentes tweets.

### 7.3.5. Motivación

La mayoría de estos pre-procesamientos fueron tomados de distintos papers [9][11][12] en los cuales se podía observar su utilidad en el contexto de aplicación. Los métodos elegidos fueron un denominador común entre las distintas bibliográficas, pero de todas maneras, se evaluó su real incidencia sobre el resultado final.

Por ejemplo, es común ver que se reemplazan los hashtags por tokens genéricos, pero a priori, hacer esto sería beneficioso en un contexto no polar (Esta afirmación se puede respaldar con [13], donde los hashtags fueron la parte clave para poder generar el corpus y se analiza su influencia sobre la determinación del sentimiento del tweet). Por ello, no todas las técnicas propuestas fueron implementadas, e incluso se agrego un feature particular que utiliza los datos sin pre-procesar (La anonimización siempre se realiza por obvias razones).

Otro pre-procesamiento no tan común que fue descartado, fue la eliminación de caracteres numéricos. Esta decisión ya es más de experiencia personal, ya que durante la etapa de obtención de tweets, se vieron varios tweets con alto contenido de caracteres numéricos, todos los cuales coincidían en neutralidad de sentimientos. Esto lleva a pensar que de alguna forma esto podría ser útil para la clasificación final.

## 7.4. Normalización de features mediante Embeddings

Dado que los features que se utilizan en la red neuronal son muchos y variados, es necesario poder acotarlos de forma tal que se pueda procesar. En el análisis previo, se señala el hecho de que la cantidad de features se ve limitada de manera nativa por el mismo entorno de twitter. Lo que no se ve limitado, es cada feature en particular. Tomando como ejemplo el caso palabras como features, las palabras no se ven muy “limitadas” en cuanto a longitud, y es necesario encontrar una representación que combine no usar memoria en exceso (como podría ser representar cada posible palabra en vectores de 140 caracteres con ceros al final para llenar), con la capacidad de tener un tamaño fijo.

Un mecanismo común para estos casos, son los denominados embeddings. Los embeddings en el campo de la matemática, son funciones inyectivas, que transforman elementos de un conjunto en elementos de otro conjunto. En este caso, se plantea mapear el espacio de las features, a un espacio vectorial. Para lograr esto se utiliza el método para generar embeddings conocido como Word2vec.

Lo primero que surge es la duda si para otras features que ya están “nor-

malizadas”, es necesario este mecanismo o se puede hacer un manejo un poco mas sencillo, como por ejemplo, matchearlas a un número. La realidad es que Word2vec es más que un simple mecanismo para transformar las features en vectores. Word2vec no solo genera representaciones en forma de vectores de las “palabras”, sino que estas representaciones permiten agrupar las mismas según significados y hasta operar con ellas de forma natural, como se puede apreciar en la figura 8.

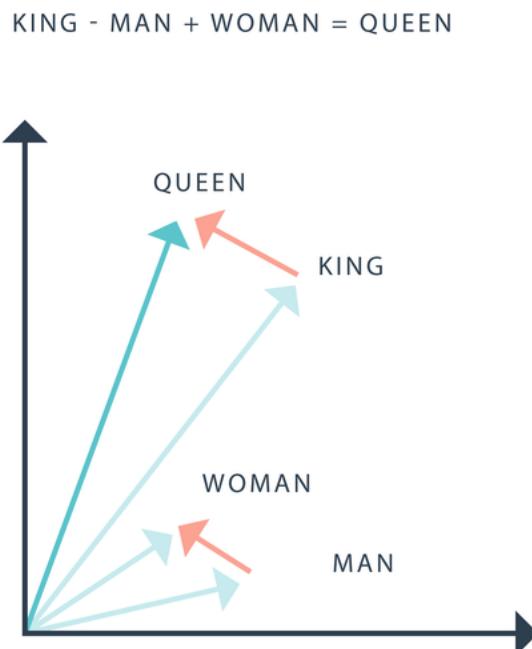


Figura 8: Word2vec: Rey - Hombre + Mujer = Reina

Esta característica del Word2Vec es muy valiosa, ya que le aporta información a la red sobre algunas relaciones entre las features. A en el ejemplo de palabras como features, aquellas que mantengan un sentido semántico similar, permanecerán juntas, es decir, tendrán representaciones similares, con lo cual la red tendría menos dificultad para encontrar un patrón general.

Esta última característica fue importante a la hora de elegir el mecanismo. Se había planteado como opción utilizar el hash trick, pero la falta de información contextual hace pensar que los resultados no serían tan buenos. Es razonable pensar que las redes neuronales en sí, son un gran resovedor de ecuaciones. Si los datos que se le ingresan se comportan más como números, los resultados serían más precisos. Siguiendo con el ejemplo de la figura 8, la red podría aprender a restar “hombre”, cosa que en un espacio generado por un algoritmo de hashing eso sería imposible.

Como su nombre lo indica, Word2vec es una herramienta pensada para funcionar con palabras. No obstante, haciendo una no muy compleja transformación de cada feature en una representación en string de la misma, se puede aplicar este mecanismo. No obstante, para cada feature en particular se deben ajustar los hiperparámetros, como es el tamaño de la ventana de contexto, ya que en algunos las features tienen como contexto “todo el tweet”, mientras que otras, las features en una vecindad acotada.

## 8. Conformacion del corpus

El corpus consiste de un juego de tweets taggeados según las 8 emociones de la clasificación de Plutwic. Estos tweets se taggean con una mecanismo crowd source en el cual a los usuarios pueden elegir en qué categoría lo clasificarán.

### 8.1. Crowdsourcing

Los mecanismos de crowdsourcing permiten reducir significativamente el esfuerzo que requiere realizar la tarea de clasificación. A su vez, permite evitar el sesgo que introduce la clasificación por parte de un solo individuo.

Este tipo de mecanismos se ha utilizado anteriormente en trabajos relacionados y es útil ya que dan cuenta de la efectividad del método, la efectividad real del mismo depende de cómo se lo aplique. Por ejemplo, en [14][15] se utiliza como mecanismo para construir un lexicon marcado según las categorías de plutzwick. Si bien los resultados fueron satisfactorios, el enfoque adoptado limita bastante la población que puede activamente colaborar con la clasificación. Dada esta restricción, se desacelera el ritmo de crecimiento de muestras clasificadas, aun utilizando herramientas de crowdsourcing pagas.

En este trabajo, se tiene un enfoque distinto para la aplicación del mecanismo. Se parte de la idea de que para cualquier usuario, es más fácil clasificar tweets completos más que solo parte de ellos. Esta afirmación se apoya en el hecho de que, durante las primeras etapas del desarrollo del sistema de tag-

ging, los usuarios dijeron que era difícil saber a que mierda correspondía cada tweet. Es razonable pensar entonces, que con menos contexto es, aun más difícil determinar las clasificaciones. Incluso, en [14] se expresa la necesidad de realizar ciertas partes de la clasificación mediante expertos en lingüística.

### **8.1.1. Definición de la validez de una clasificación**

A decir verdad, no hay una única respuesta o una respuesta correcta a la hora de determinar el sentimiento de un tweet. Lo más cercano a esto podría ser como lo identifique su propio autor. No obstante, esto puede no coincidir con la valoración que tiene del mismo la mayoría. Un tweet no tiene valor como la clasificación de una sola persona particular, sino como la valoración de varios, ya que en si, el sistema intenta poder imitar el razonamiento de la mayoría, y no de solo una persona en particular.

## **8.2. Corpus utilizados como base**

Dados algunos resultados que surgieron de las pruebas manuales realizadas, se optó por comenzar utilizando un dataset que estuviera un poco mas maduro. Lamentablemente no hay disponible un dataset que abarque completamente las ocho emociones de la rueda, debido a dos factores: Por un lado, no son tan comunes de encontrar, ya que esta demostrado que se es más tendiente a usar twitter feliz que de otra forma [13][14][16]. Por otro lado, las otras emociones son mucho más complejas de identificar, y en algunas pruebas con usuarios cualesquiera, era bastante común encontrarse con la pregunta de: “Como puedo identificar esto”.

### **8.2.1. Corpus en inglés**

Se utilizó el corpus desarrollado por Saif Mohammad [17] que posee algunas de las clasificaciones utilizadas en este trabajo.

### **8.2.2. Corpus en español**

Para la parte de datos en español, se utiliza como corpus base creado para el trabajo [13]. El dataset publicado[18] si bien se ofrece como un corpus balanceado, los tweets no están directamente disponibles, con lo cual muchas muestras se pierden al ser imposible poder recuperarlas<sup>4</sup>.

---

<sup>4</sup>A esto igual hay que sumarle el pobre empeño que hubo en la conformación del set publico. Algunos de los IDs carecían de todos los dígitos necesarios.

## 9. Pruebas preliminares del modelo

El modelo final utilizado surgió en parte de los resultados de las distintas pruebas que se fueron realizando sobre algunas implementaciones anteriores. A modo de resumen, se detallan algunas pruebas importantes y su impacto sobre el modelo final.

La primera versión se hizo utilizando el dataset construido mediante la herramienta de tagging y otros datasets disponibles públicamente en la red. Se utilizó un esquema sencillo 3,5,7,7 y solo los tokenizers de palabras, caracteres y caracteres sin procesar. Se realizaron 80 iteraciones de entrenamiento, logrando aproximadamente un 0.74 de éxito en la clasificación del set de entrenamiento. Sin embargo, no logró jamás superar el 0.35 de éxito en las clasificaciones del set de pruebas, con picos máximos de 0.41. Suponiendo que podría ser un error en cuanto a la elección del set de prueba, se montó el clasificador en el sistema completo y se lo dejó correr para clasificar tweets disponibles en el stream. Al ser la mayor parte del dataset en inglés, se filtraron los tweets de solo este idioma. Los resultados en un contexto real, fueron aun menos satisfactorios que sobre el set de pruebas, incluso para las categorías que tenían un mayor numero de muestras. Este comportamiento se atribuyó a las grandes discrepancias en el volumen de datos de cada categoría.

Para probar si el dataset era el problema, se realizó una prueba utilizando únicamente los tweets provenientes del dataset de EmoInt. Se realizó bajo las mismas condiciones que la prueba anterior, solo limitando el dataset. Se logró un 0.95 sobre el set de entrenamiento y aproximadamente un 0.7 en el set de prueba. El set de EmoInt no presenta las 8 emociones descritas por Plutchik, solo 4, lo que hace que cualquier tweet que presente alguna de las 4 restantes sea mal clasificado. No obstante, el hecho de que las clasificaciones mejoraron dramáticamente, aun cuando se redujo el tamaño del set de datos, sugirió que el problema radicaba realmente en la baja cantidad de muestras para algunas de las categorías.

En este punto, las iteraciones de entrenamiento tardaban realmente mucho, lo que motivó a buscar alternativas a la metodología aplicada para poder acelerar el proceso. Por un lado se comenzó a guardar las representaciones matriciales de cada tweet, ya que era un proceso caro y que se realizaba de forma constante. Por el otro, se buscó modificar el esquema de la red para poder tener menos que entrenar.

Luego de los cambios, se re-entrenó y se evaluó en condiciones reales el

modelo generado. Los resultados no fueron para nada satisfactorios pese a haber logrado aprender bien en el set de entrenamiento. Varias revisiones mas tarde, se llego a identificar que el problema radicaba en el dataset utilizado. El mismo esta armado de forma tal que cada categoría contiene tweets marcados con la probabilidad de pertenecer a la misma, lo que no se detecto antes fue que esa probabilidad podría ser 0.

Posteriores entrenamientos se realizaron sobre un corpus ya mucho mejor formado, únicamente conteniendo tweets en inglés. Ahora se pudo realizar el ajuste fino de los hiperparametros, ya que los resultados eran ahora si satisfactorios en el contexto real. Se pudo reducir el tamaño de los embeddings a nivel caracter, sin perder precisión, y se agregaron mas filtros al stream de palabras mejorando la precisión en el set de testing.

Con la red funcionando para el idioma inglés, se procedió a incorporar el corpus en español sin variar ningún parámetro. Contrario a lo pensado en un primer momento, la red continuo funcionando sin notables reducciones en la precisión de las clasificaciones. Por ello, en vez de agregar mas features de soporte, se comenzó a ya realizar la configuración fina del esquema.

## 10. Implementacion

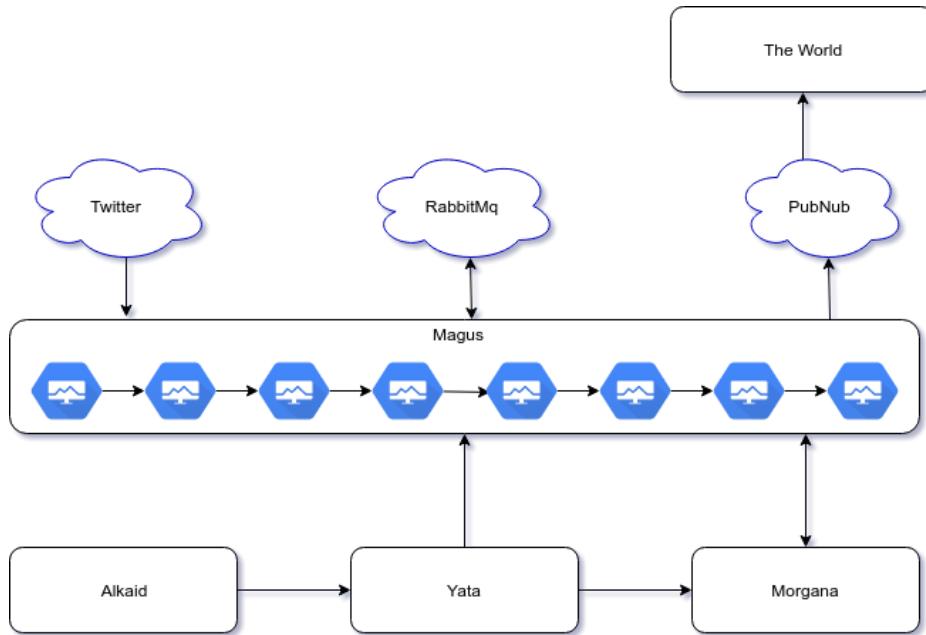


Figura 9: Entorno de Magus

### 10.1. Magus (Pipeline)

Magus esta pensado como un sistema de tiempo real, que pueda alimentarse del stream de twitter y realizar la clasificación de forma casi instantánea. Para lograr este efecto, es necesario que el procesamiento de cada tweet sea rápido e independiente del procesamiento de los otros. La forma de lograr esto es mediante una arquitectura tipo pipeline. En este tipo de arquitecturas, el procesamiento se realiza de forma concurrente, y es muy común de ver en microprocesadores como forma de poder mejorar el uso de los recursos de hardware. En este caso, esta estrategia busca mejorar la distribución del uso del CPU.

El objetivo principal del pipeline es mitigar el stress que implica preprocesar cada tweet para poder extraerle las features, y mismo el proceso de extracción. En muchos casos, el preprocesamiento se puede hacer de manera tal, que ubicado correctamente, sirva por igual a todo un set de tokenizers<sup>5</sup>. Si estas operaciones no insumieran un tiempo no despreciable para este trabajo, probablemente esta arquitectura no seria necesaria en lo absoluto,

<sup>5</sup>Los tokenizers son las partes encargadas de extraer cierto tipo de features del tweet.

A modo de seguir una convención de nombres acorde al nombre del proyecto, las distintas partes del pipeline poseen nombres relacionados.

### 10.1.1. Kuhn (Master)

Kuhn es la parte encargada de iniciar los cores. Los mismos los crea en base a un archivo de configuración en formato json con la siguiente estructura:

```
"<core name>": {  
    "tag": "<core tag>",  
    "replicas": "<replicas>",  
    "input_queue": "<queue name>",  
    "output_queue": "<queue name>"  
},  
[...]
```

**Core Name** : Nombre con el que se identifica al core al momento de mostrar los mensajes de logging. Cada tipo de core debe tener un nombre distinto.

**Core Tag** : Identifica el tipo de core al que hace referencia. La correspondencia entre los tags y los constructores de las clases cores específicas, se encuentran definidos en el código de Kuhn.

**Replicas** : Cantidad de cores iguales (procesos) que se lanzan para esta configuración.

**Queue Name** : Nombre de la cola de la cual se leen o en la cual se escriben los datos según corresponda. Los nombres definidos son arbitrarios, y solo sirven para diagramar como se conectan los cores entre si.

### 10.1.2. Magus Core (Worker)

Los cores representan cada una de las etapas del pipeline. Cada una de estas etapas son independientes y están interconectadas entre si. Esto es, que cada etapa procesa un input que puede provenir o no de otro core, pero que a fines prácticos solo requiere que el input contenga la información necesaria.

Cada core está implementado como una clase de Python, y cumplen cada uno solo con una función específica. Todos los cores leen datos de una cola, y escriben los resultados en una cola de salida, con excepción de los cores productores (generan datos) y los cores emisores (envían los resultados por algún canal de broadcast).

### 10.1.3. Modelo distribuido

El hecho de que la arquitectura conste de varias etapas interconectadas e independientes entre sí, permite que se pueda extender fácilmente para aumentar el poder de procesamiento. Mas aun, dado el mecanismo de conexión entre las mismas, se puede extender el sistema de forma distribuida en una red local o incluso Internet. En la figura 10 se puede apreciar como cada core en particular corriendo sobre workers distribuidos se conectan entre si mediante RabbitMq.

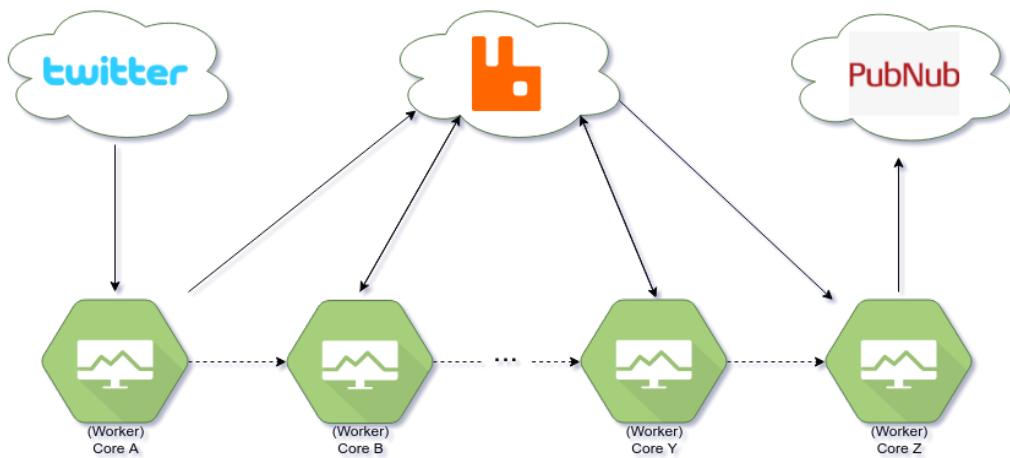


Figura 10: Esquema del modelo distribuido

Este tipo de enfoques se han utilizado ya en otros ambientes de investigación, como por ejemplo para procesar imágenes del espacio, ayudar a procesar el genoma humano o generar nuevos modelos para la predicción del clima [19]. En este caso, cada cliente puede levantar un subset de cores y unirlos a la red. Excede el scope de este trabajo, así que se deja a futuro implementar un mecanismo que pueda de forma autónoma, asignar cores a los clientes según las necesidades del sistema y no de forma arbitraria.

Para facilitar la distribución del sistema, se utiliza docker. Docker permite poder levantar de forma sencilla, el entorno de ejecución de los cores. De esta forma, se evita la necesidad de instalar y configurar las herramientas necesarias para poder tener todo funcionando, solo tener instalador docker.

### 10.2. Morgana (Clasificador)

Un punto clave del modelo de procesamiento propuesto, radica en la capacidad del sistema de poder procesar múltiples tweets de similar manera.

Para garantizar esto, se utiliza tensorflow model server, que permite acceder al clasificador por medio de requests y todas sobre el mismo modelo. Esto permite por un lado poder garantizar que todos los cores de clasificación tengan acceso al mismo modelo y por el otro, da la libertad de levantar el servicio en una infraestructura acorde a las necesidades del mismo.

A la combinación entre el servidor y el modelo se lo denomina Morgana. Para acceder al servidor, se utiliza la biblioteca de RPC para Python [46], que permite enviar y recibir las requests. La única desventaja, es que esta biblioteca solo funciona con Python 2, con lo cual, los cores de clasificación deberán ser ejecutados en una versión distinta de Python al resto de los cores (que aprovechan las características nuevas implementadas en Python 3).

### 10.3. Alkaid (Web Tagger)

Como ya se explico anteriormente, Magus se apoya en un modelo del tipo crowdsource para poder generar de forma dinámica el conocimiento. Para ello es necesario tener alguna herramienta que permita a los usuarios que quieren colaborar con el proyecto, hacerlo de forma fácil y sencilla. A dicha herramienta se la denomino Alkaid.

Alkaid es una simple aplicación web desarrollada utilizando Python3[40] y Jinja2[49]; la cual se encuentra hosteada en el servicio gratuito de Heroku[58] y permite realizar tres operaciones básicas:

- Agregar nuevos tweets
- Clasificar tweets existentes
- Validar tweets pre-clasificados

Cada tipo de clasificación, impacta de forma diferente sobre la base de conocimiento. Las clasificaciones más específicas se ven más beneficiadas que otras mas genéricas o personales. Por ejemplo, no es igual clasificar si cierto tweet encaja en cierta categoría que decidir completamente a que categoría pertenece. De la misma forma, la clasificación de un nuevo tweet agregado agregado por el usuario, tiene mayor peso, ya que supone que el usuario tiene un mayor nivel de confianza en la misma. Cada clasificación se guarda en la base de datos como una cantidad de “puntos” asignados a cada categoría en particular, y la clasificación final se decide en base a cual es la categoría que tiene mayor porcentaje. Los distintos niveles de confianza en las clasificaciones, entonces se traducen en mayores o menores puntajes. La clasificación de validación por su parte, son las únicas que tienen la capacidad de restar puntos, ya que responde a la pregunta “Esta es una clasificación valida?”

más que “A que clasificación pertenece”, porque la segunda pude tener más de una respuesta válida.

Alkaid está soportada de fondo por una base de datos SQL, hosteada en el servicio gratuito de ElephantSQL[59]. En la misma se almacenan los resultados de las clasificaciones y al igual que Yata, accede mediante la biblioteca SQLAlchemy[47] de Python.

Se diseñaron varias alternativas del frontend hasta llegar a la versión actual. Se decidió utilizar el diseño de la rueda de plutchwik acompañada de las ilustraciones porque resultaba no solo fácil de usar sino también visualmente intuitivo de que hacer. Versiones anteriores sufrían de una mala usabilidad y la respuesta de aquellos que lo testearon fue no positiva. Los mismos, presentaron buenas críticas respecto a la usabilidad de la versión actual. La versión en español de la rueda es una adaptación de la versión original de la imagen en inglés.

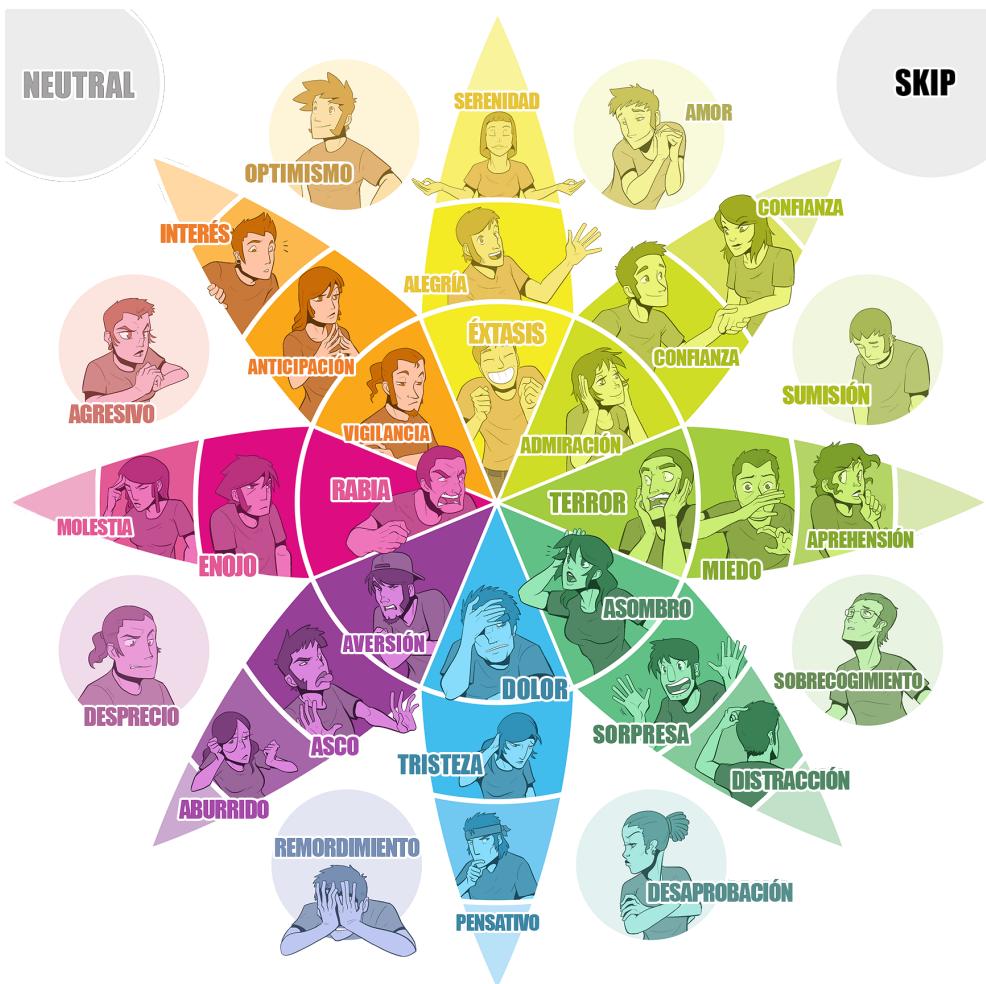


Figura 11: *Plutchik wheel of emotions*

#### 10.4. The World (Mapa en tiempo real)

Como contexto de aplicación del sistema, se propuso mostrar los resultados en tiempo real sobre un mapa interactivo. Como el objetivo del mapa es ser lo suficientemente intuitivo de entender, se realizó un denominado heatmap.

Los heatmaps o mapas de calor, son formas de presentar los datos, en los cuales las intensidades de los valores de los datos se presentan asociadas a un color. Al igual que los sensores térmicos, colores más brillantes representan mayor intensidad en los datos, y viceversa. Este tipo de visualizaciones cumplen la condición de ser intuitivas y fáciles de interpretar.

Para este trabajo, el mapa se implementó como un mapa multicapa, don-

de cada capa es un heatmap asociado a una de las categorías de clasificación. En vez de utilizar el esquema clásico de colores, a cada categoría se le asigna una paleta de colores distinta, siguiendo con la convención de oscuro a claro dependiendo de la intensidad. De esta forma, se pueden visualizar en simultáneo todas las categorías sin perder noción de que se está viendo. Las paletas del heatmap matchean a los colores utilizados en el frontend web del tagger (figura 11).

The World fue desarrollado como un frontend web, combinando una página base en HTML y funciones en JavaScript. Para el mapa se utilizó la API de Mapbox[50], para la conexión del stream PubNub[54] y para servirlo Flask[48] mas Gunicorn[53]. The World al igual que Alkaid se encuentra hosteado en el servicio gratuito de Heroku[58], lo que permite acceder a él desde cualquier lado sin necesidad de tener ningún tipo de infraestructura particular.



Figura 12: Mapa generado por 'The World'

## 10.5. Yata (Trainer)

Yata es la parte involucrada en el entrenamiento del modelo. El mismo se desarrolla en tres etapas:

- Generación de embeddings

Se generan los archivos de vectores (embeddings) para cada grupo de features utilizando word2vec. Dado que no todas las muestras disponibles se encuentran clasificadas, realizar la generación de las features en una etapa distinta a la de entrenamiento, permite crear embeddings mas representativos.

- Generación de los batches para training y testing

La extracción de features y posterior matcheo de los mismos para cada tweet es una tarea costosa y repetitiva. Ya que el volumen de datos no permite mantenerlos en memoria durante todo el proceso de entrenamiento, en esta etapa se generan los batches y se persisten en disco.

- Entrenamiento del modelo

Luego de la ejecución de Yata, el modelo se exporta para poder ser levantado por el servidor de tensorflow; ademas de los archivos con los embeddings para cada feature, a ser utilizados en el pipeline.

## 11. Evaluación del modelo

### 11.1. Primera evaluación

A continuación se muestran los gráficos de los resultados del entrenamiento del modelo propuesto durante 25 epochs.

### 11.1.1. Accuracy

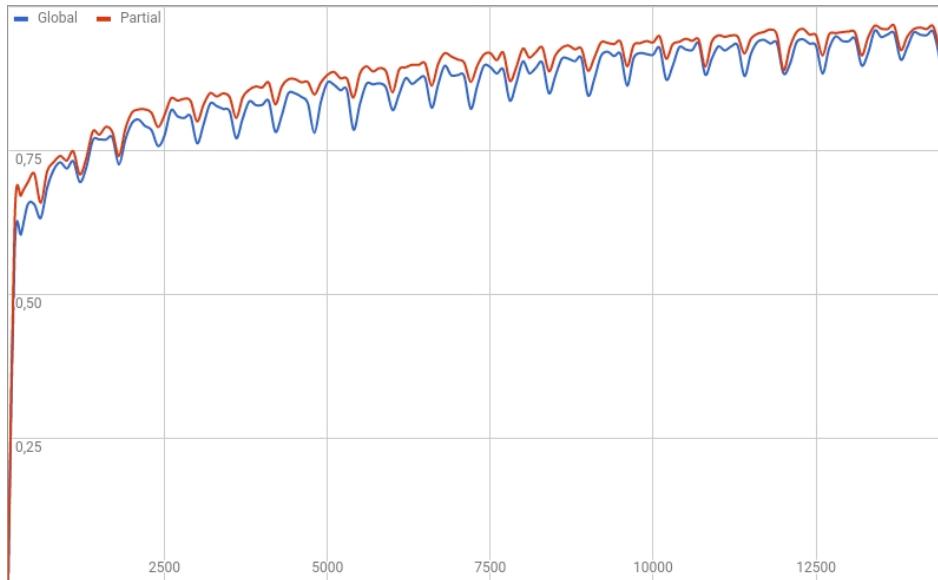


Figura 13: Precisión del clasificador para la salida “Global” y la salida “Partial” en el set de training.

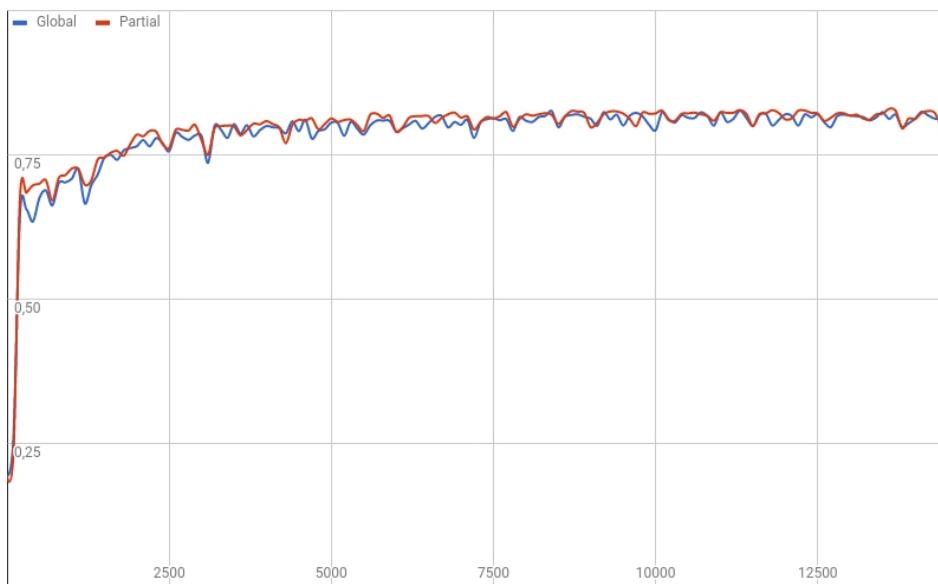


Figura 14: Precisión del clasificador para la salida “Global” y la salida “Partial” en el set de testing.

Se puede ver que tanto en el set de training como en el set de testing, la salida “Partial” tiene en general mejores resultados que el la salida “Global” (es mucho mas notable en el set de testing). En el set de training ademas, se pueden apreciar algunas caídas de precisión periódicas. Se atribuye la causa de esto al cambio de trainer que ocurre también de forma periódica. Esto hace pensar que este uso de trainers globales, no contribuyen de forma positiva al entrenamiento de la red total.

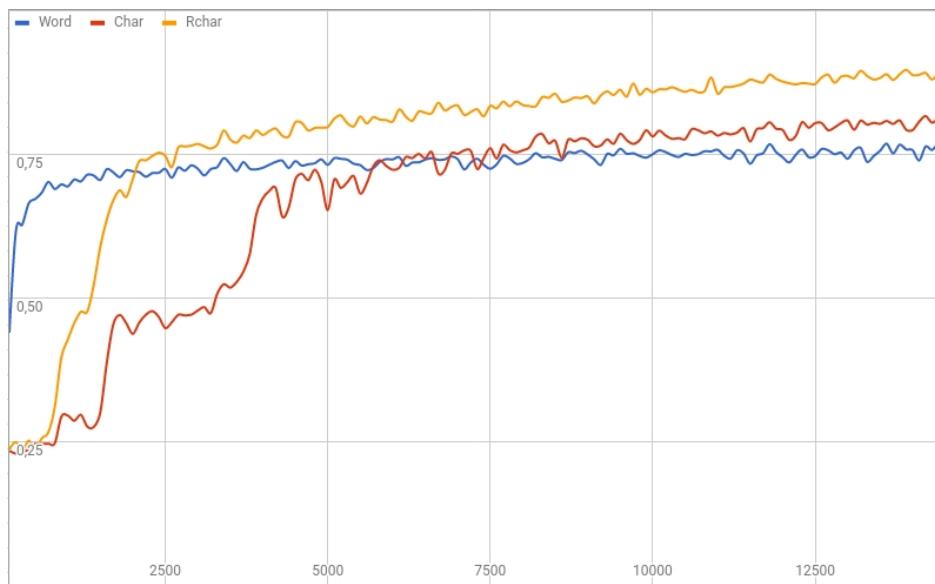


Figura 15: Precisión del clasificador para los streams en el set de training.

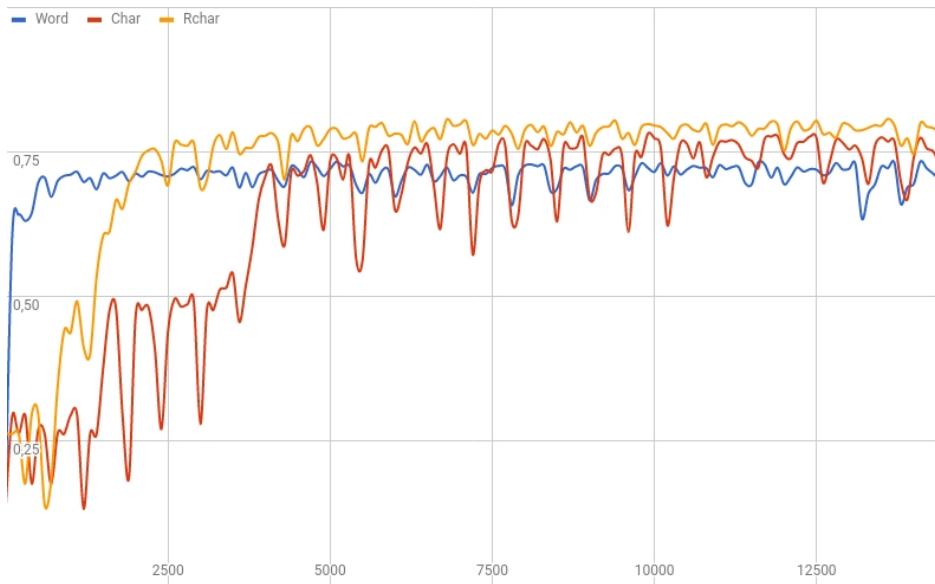


Figura 16: Precisión del clasificador para los streams en el set de testing.

De los gráficos de precisión para cada uno de los streams se desprenden las siguientes cuestiones importantes. En primer lugar, que el stream que tiene como features a las palabras, converge rápidamente a lo que es su valor máximo y prácticamente no mejora su precisión con el correr de las iteraciones. En segundo lugar, que los streams basados en caracteres, si bien tardan mas en dar buenos resultados, en general tienen mejor performance que el stream de palabras. Por ultimo, en comparativa, el stream de caracteres sin procesar, es mucho mas estable que el de los caracteres pre-procesados (en particular en el set de testing).

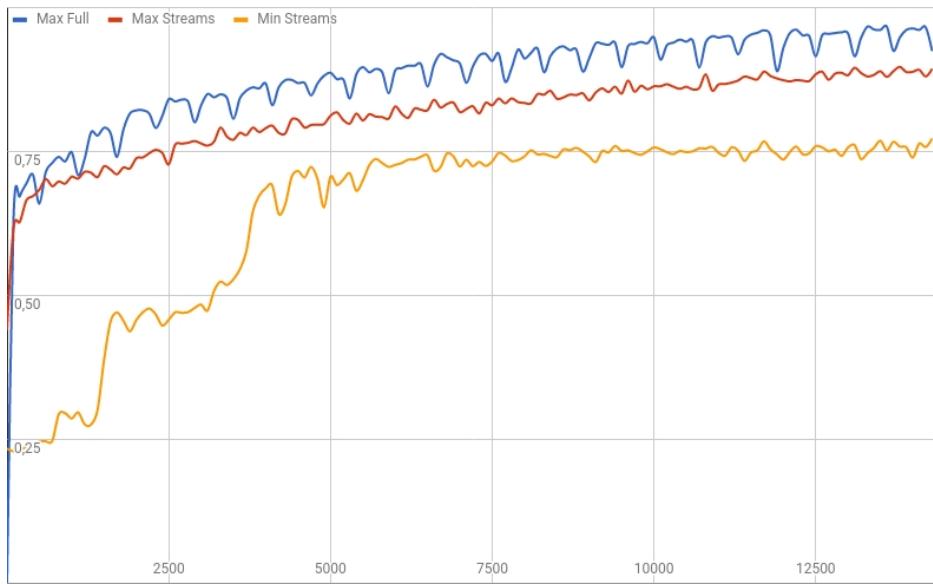


Figura 17: Comparación de precisión entre los streams de salida con los streams internos en el set de training.

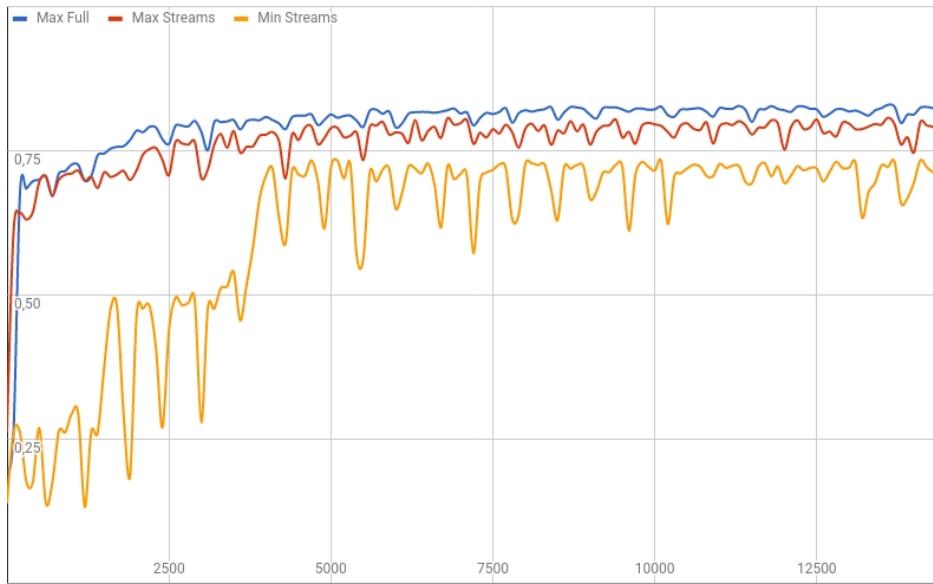


Figura 18: Comparación de precisión entre los streams de salida con los streams internos en el set de testing

Si se comparan los mejores resultados de los streams de salida contra los mejores resultados de los streams internos, se puede ver claramente como

los primeros siempre tienen una mayor precisión; lo cual implica que aunque sea levemente, este enfoque mejora cualquier resultado que se pudiera tener utilizando solo un tipo de feature.

### 11.1.2. F1-Scores

Para que el clasificador funcione de forma aceptable se debe garantizar que la dispersión entre los “F1 Scores” entre las clases sea mínima (fig. 19/20), y que el “F1 Score” promedio sea alto (fig. 21/22).

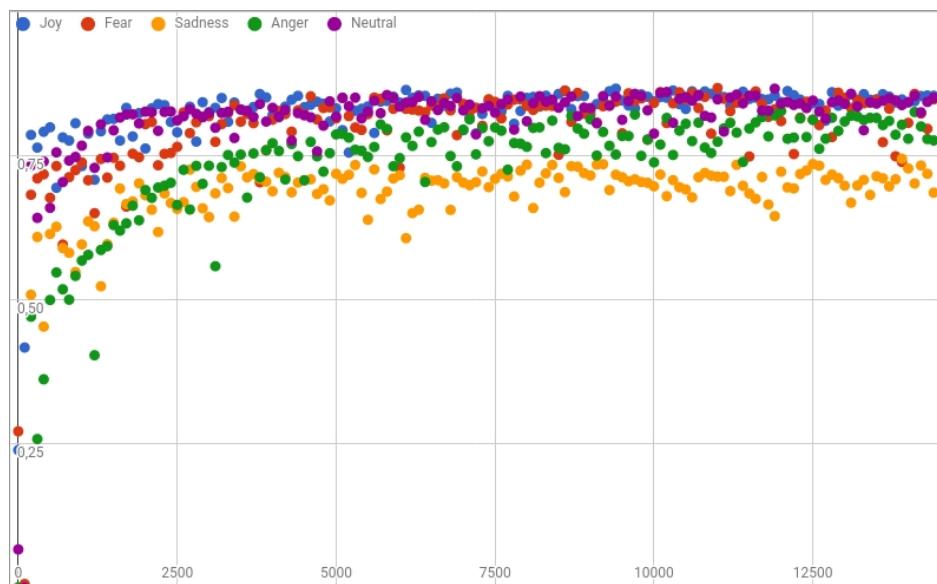


Figura 19: Dispersión de los “F1 Scores” para cada clase en el stream “Global”

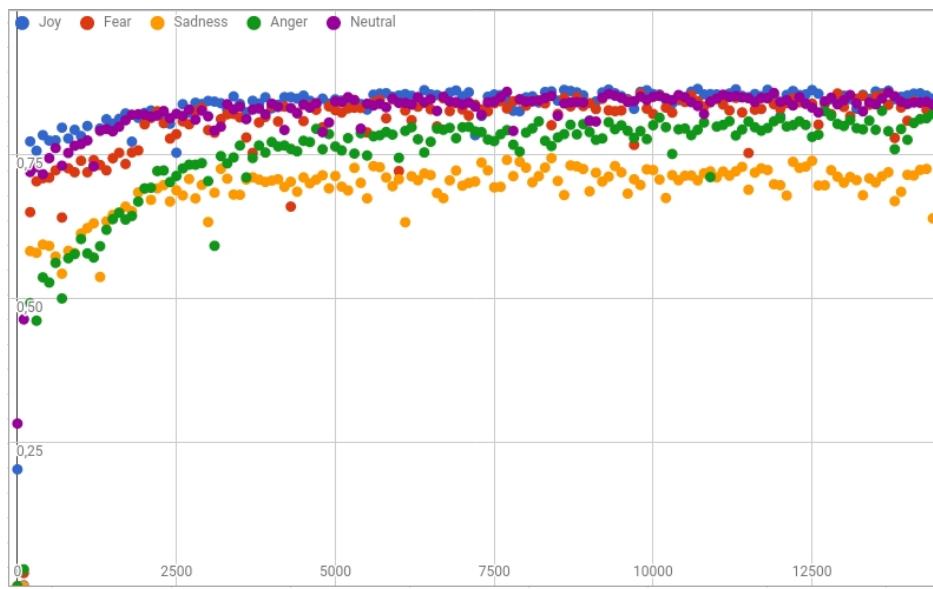


Figura 20: Dispersión de los “F1 Scores” para cada clase en el stream “Partial”

Se puede apreciar que la salida “Partial” tiene una mucho menor dispersión de los resultados, entre cada clase y entre las diferentes iteraciones para cada clase en particular. Estos resultados combinados con los de la sección anterior, dejan ver que no tiene sentido continuar utilizando los 2 streams de salida ya que uno es claramente mas efectivo que el otro.

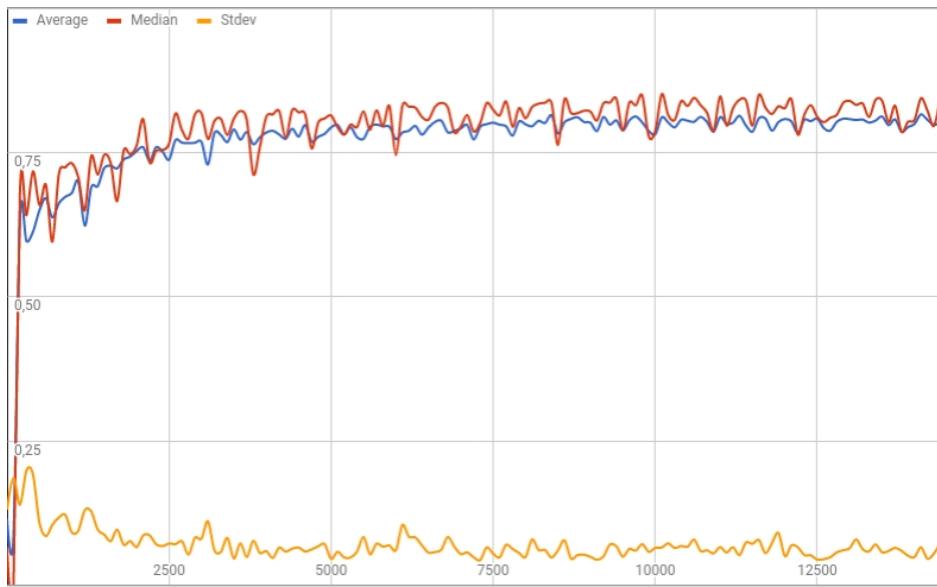


Figura 21: Estadísticas de los “F1 Scores” para cada clase en el stream “Global”

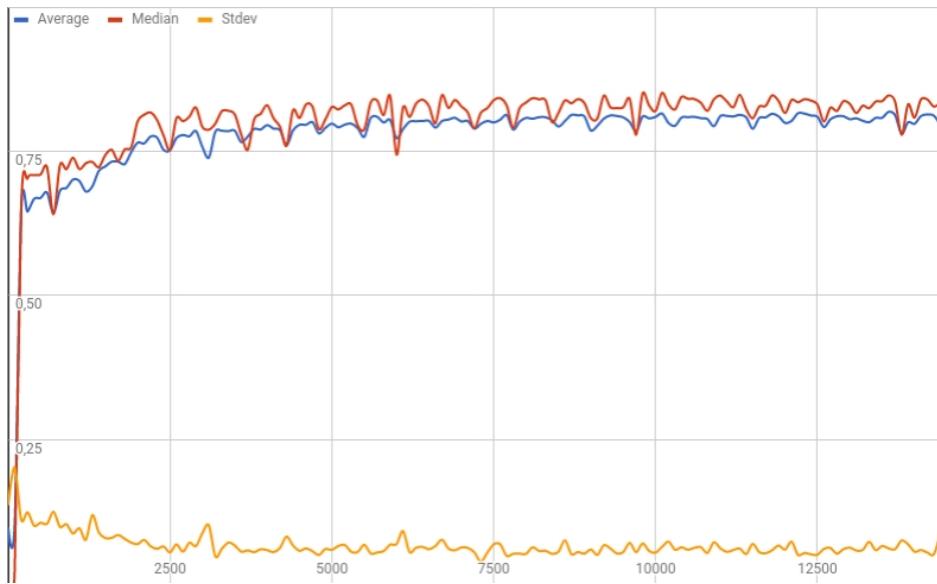


Figura 22: Estadísticas de los “F1 Scores” para cada clase en el stream “Partial”

De las estadísticas se puede ver como la diferencia entre los F1-Score de cada clase son mucho menos variables en el segundo stream, y que ademas, los

valores de la mediana para este ultimo suelen superar el valor del promedio, lo cual es un buen resultado.

## 11.2. Evaluación final

Para la evaluación final del modelo, se utilizaron los datasets de la prueba anterior, sumado a las muestras recolectadas por el web tagger. Las muestras obtenidas del tagger fueron filtradas por lenguaje, dejando solo las marcadas como español o inglés (misma lógica de filtrado que aplica el pipeline a al inicio). De estos, la totalidad se utilizó para la construcción de los embeddings. Esto por un lado permite tener una idea un poco más realista de como funciona el sistema en producción. Por el otro lado, armar los embeddings con un número más grande y variado de muestras (suponiendo que se generan correctamente), permite poder realizar clasificaciones más realistas en elementos que escapan los sets de testing y training.

Para esta evaluación, la configuración que se utilizó fue:

- Stream de palabras:
  - Filtros: [1,3,5,7,9,9]
  - Cantidad de filtros: 200
  - Cantidad de hidden neurons: 1024
  - Tamaño de embeddings: 300
- Stream de caracteres:
  - Filtros: [3,5,7,7]
  - Cantidad de filtros: 200
  - Cantidad de hidden neurons: 512
  - Tamaño de embeddings: 150

### 11.2.1. Accuracy

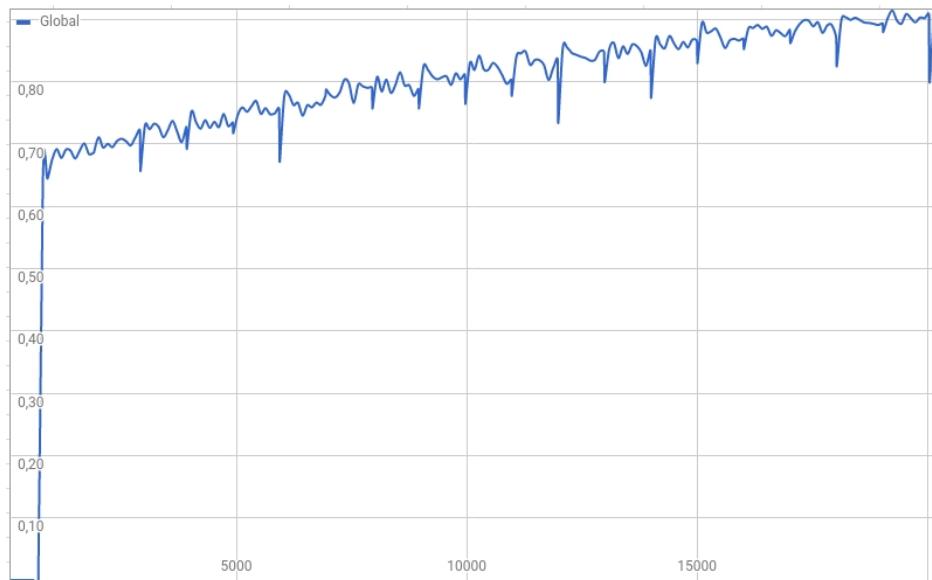


Figura 23: Precisión del clasificador para el set de training.

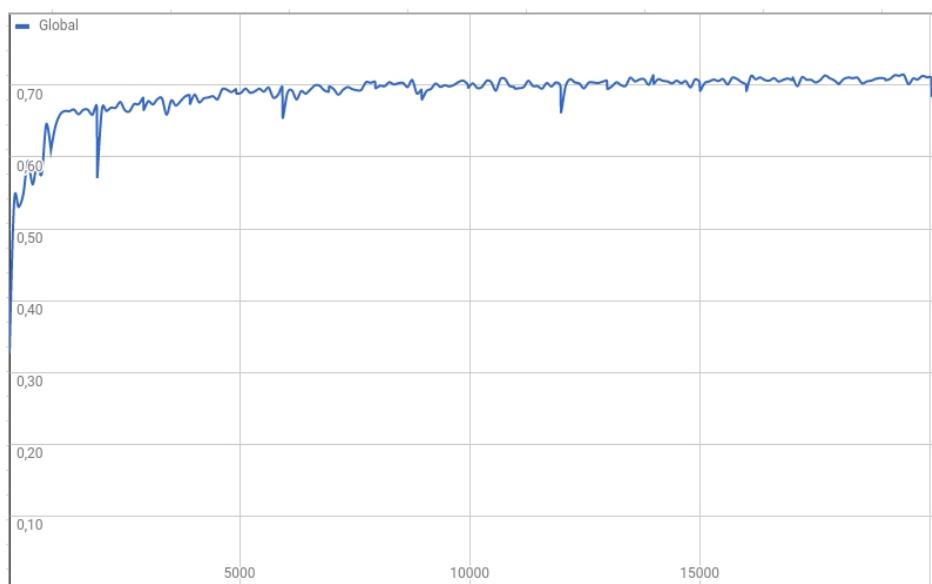


Figura 24: Precisión del clasificador para el set de testing.

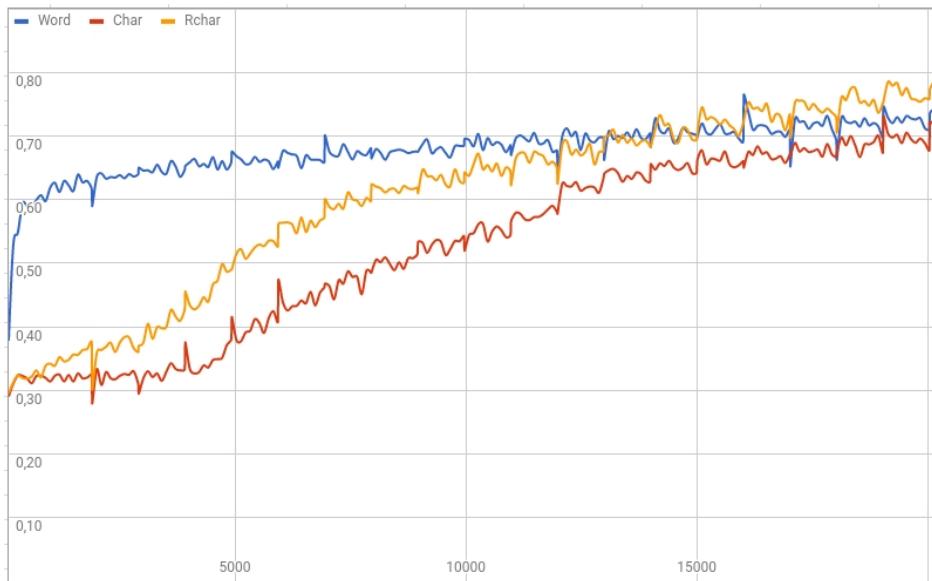


Figura 25: Precisión del clasificador para los streams en el set de training.

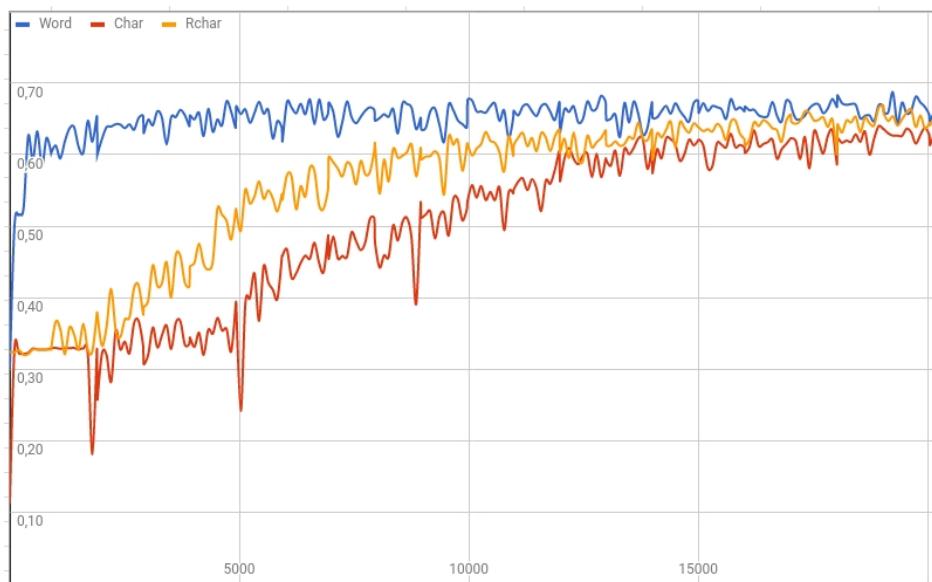


Figura 26: Precisión del clasificador para los streams en el set de testing.

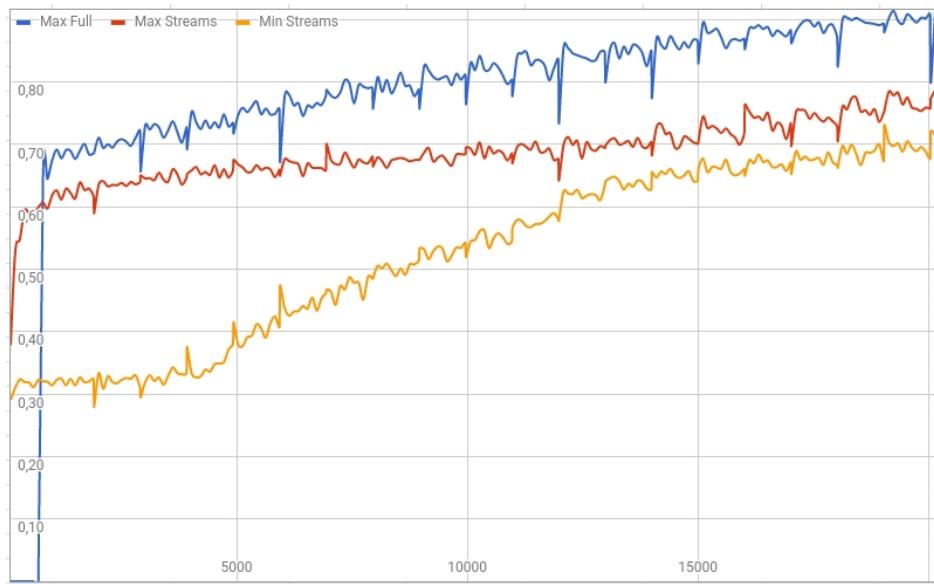


Figura 27: Comparación de precisión entre los streams de salida con los streams internos en el set de training.

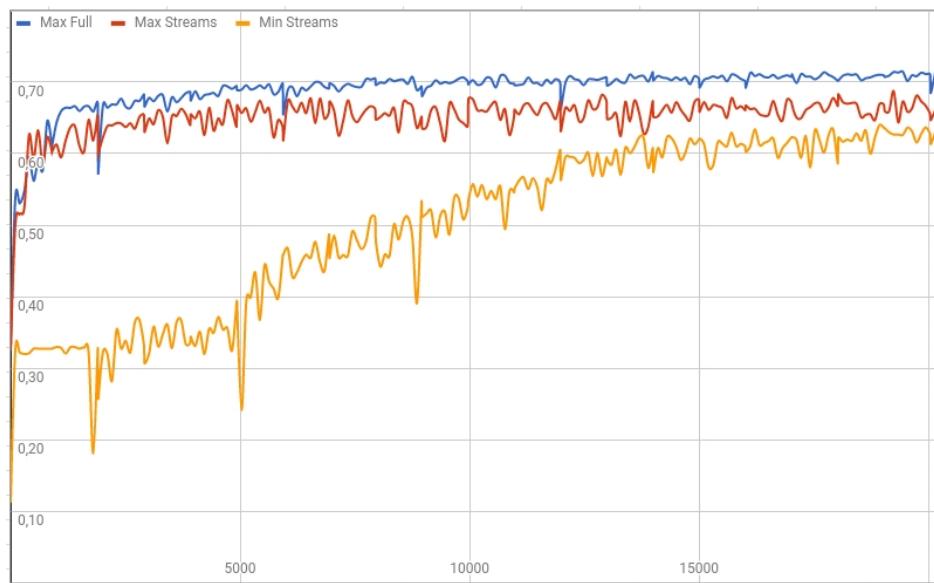


Figura 28: Comparación de precisión entre los streams de salida con los streams internos en el set de testing

### 11.2.2. F1-Scores

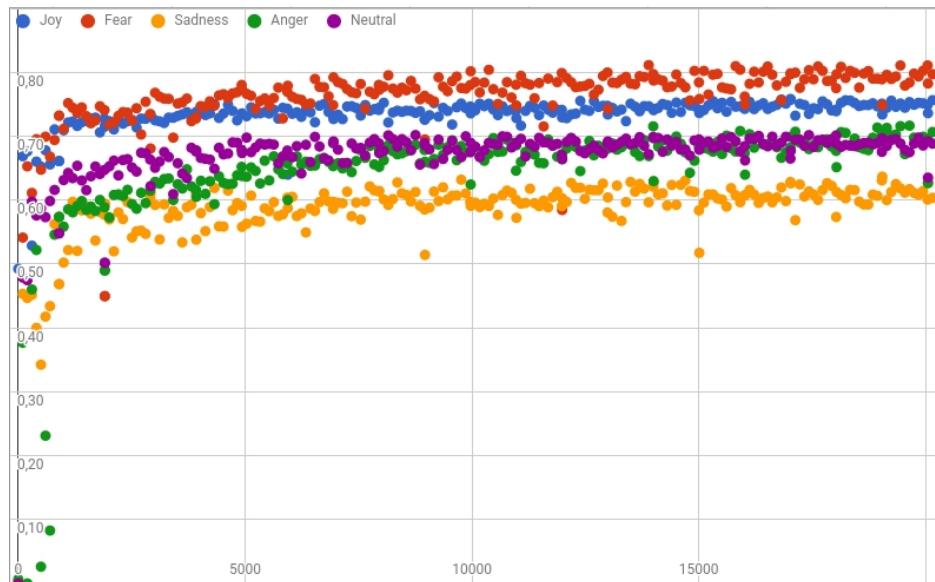


Figura 29: Dispersión de los “F1 Scores”

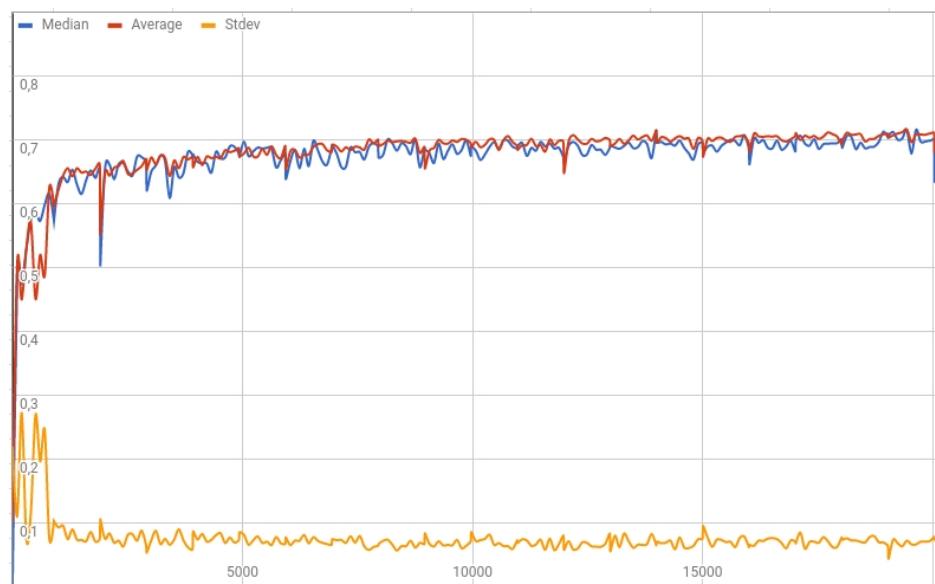


Figura 30: Estadísticas de los “F1 Scores”

Para las pruebas en un entorno mas realista, se puede observar que el clasificador se comporta de manera similar a lo visto en el ejemplo de labora-

torio<sup>6</sup>. La inclusión de nuevos tweets tanto para la generación de embeddings como para el entrenamiento ralentizaron el proceso de aprendizaje de los streams de caracteres, pero siguen cumpliendo que a la larga logran similares o iguales resultados parciales que el stream de palabras.

## 12. Conclusiones finales

Si bien el trabajo fue un desarrollo, requirió de bastante investigación en los temas relacionados y probar distintos enfoques en un nuevo campo de aplicación. Como aun no hay mucho desarrollado al respecto, resulta de interés plasmar algunas de las lecciones aprendidas durante el desarrollo.

En primer lugar, se planteo como objetivo buscar una alternativa a los clasificadores polares clásicos, que ademas, funcionara en un ambiente multilenguaje. Se puede suponer que el stream de palabras es en gran parte un clasificador polar clásico al que se le cambio simplemente el tamaño de la salida y la configuración de los filtros, y así usarlo como punto de comparación. De esto se desprende que entonces los clasificadores clásicos basados en CNN tienen la capacidad de funcionar de manera aceptable<sup>7</sup> en un entorno multivaluado con algunas simples modificaciones, contrario a lo que se creía en un primer momento. No obstante se requiere de mas que ello para poder tener resultados significativamente buenos.

Una conclusión interesante que se puede sacar observando los gráficos, en particular las figuras 26 y 28, es que la red se apoya fuertemente en los resultados del stream de palabras. Se puede apreciar como la precisión del clasificador sigue una tendencia de crecimiento (en el set de testing) similar a la de la presionen del stream de palabras. Se puede entonces inferir que, al menos con la evidencia de las pruebas realizadas, el tipo de feature mas relevante para las clasificaciones de este tipo, continúan siendo las palabras; mientras que otros tipos de features pueden servir de soporte para mejorar esta clasificación base.

Por ultimo, se pudo apreciar de primera mano el efecto de dataset en el resultado final del clasificador. Fue sorprendente el hecho de que la red lograse encontrar patrones para poder clasificar correctamente un dataset completamente mal formado, pese a no tener ninguna clase de coherencia. En este caso, los resultados erróneos pudieron recién visualizarse al finalizar el largo proceso de entrenamiento y fue difícil localizar la causa. Es por esto, que

---

<sup>6</sup>Decimos ejemplo de laboratorio porque los corpus utilizados están mucho mas normalizados que el promedio de los tweets en la realidad

<sup>7</sup>Precisión por encima de 0.6 para 5 clases cuando el baseline random es de 0.2

a modo de conclusión final, se puede decir que dedicar mayor esfuerzo a la determinación de como conformar el dataset en las primeras partes de un proyecto, puede evitar muchísimos problemas luego, mas de los que uno a priori se podría imaginar<sup>8</sup>.

## 13. Trabajo a futuro

Dado el scope del presente trabajo, algunas cosas fueron dejadas de lado al no ser esenciales al objetivo planteado. Se detallan a continuación, algunos puntos sobre los cuales se puede continuar trabajando:

Profundizar en los mecanismos del tipo ensamble: El esquema de red que utiliza Morgana se puede considerar como un ensamble de varias redes neuronales. En este trabajo, solo se analizo utilizando tres redes similares entrenadas para cumplir con el mismo objetivo. En [6] se utilizan varios tipos distintos de clasificadores intermedios, y se analizo como posibilidad agregar un clasificador binario (positivo/negativo), pero no se termino agregando dado:

- Mientras mas compleja sea la red, mas lento es el proceso de clasificación, lo que no es una característica deseada.
- Los costos en tiempo que implican agregar una nueva red, considerando que el proceso de entrenamiento en el equipo de desarrollo insumía días<sup>9</sup>.

Mejorar el sistema de tagueo online: Alkaid funciona bajo la premisa que las clasificaciones se ingresan a conciencia y con buena voluntad. En un contexto mas realista, es muy propensa a ataques tal y como esta construida, por lo cual debería de incorporarse un mecanismo que pueda de forma autónoma distinguir entre buenas clasificaciones de spam. Esto no se implemento ya que el objetivo del trabajo no era este, y Alkaid fue desarrollada únicamente como una herramienta soporte.

Mejorar el sistema de almacenamiento temporal: Los cores de Magus vuelcan la información temporal en un directorio compartido por todos. Esto luego es levantado para realizar las clasificaciones. El problema

---

<sup>8</sup>A modo de nota al margen, siempre se trabajo pensando en que si el dataset estaba mal, se evidenciaría durante el entrenamiento mediante la incapacidad del modelo de aprender.

<sup>9</sup>El proceso en promedio tomaba 2 días, dado que no se disponía de una placa de vídeo compatible con CUDA.

radica en que la implementacion actual requiere de un volumen compartido entre todas los workers del pipeline, que bien podría ser una unidad de red, pero no es una solución que escale fácilmente si se distribuye a computadoras particulares. Una solución a esto podría ser migrar ese mecanismo de almacenamiento a un almacenamiento del tipo NoSQL como Cassandra. Esto no se implemento ya que a fines prácticos, se puede realizar un deploy distribuido en una red local.

Mejorar el mecanismo de re-entrenamiento: Al igual que el item anterior, Yata no tiene la capacidad de operar completamente a travez de Internet. Yata esta preparado para poder ser ejecutado como un proceso cron cada x cantidad de tiempo, pero carece de la capacidad de reiniciar a Morgana con los cambios si es que esta no se encuentren en el mismo ordenador, y por ello requiere de un reinicio manual por parte del usuario. Al igual que antes, la capacidad de poder hacer un deploy en una red local hacen que no se viera la necesidad de implementar este mecanismo de forma urgente.

# Referencias

## Bibliográfica consultada

- [1] *Rae.* Diccionario de la lengua española.  
<http://dle.rae.es/>
- [2] *The Tanl Pipeline*  
G. Attardi, S. Dei Rossi, M. Simi. The Tanl Pipeline. Proc. of LREC Workshop on WSPP, Malta, 2010.  
<http://pages.di.unipi.it/attardi/Paper/LREC10%20Pipeline.pdf>
- [3] *Sentiment Analysis: Beyond Polarity (Thesis Proposal)*  
Phillip Smith October 2011.  
<https://www.cs.bham.ac.uk/~pxs697/publications/documents/rsmg3.pdf>
- [4] *Recent Trends in Deep Learning Based Natural Language Processing*  
Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria, Junio 16-17, 2016.  
<https://arxiv.org/pdf/1708.02709.pdf>
- [5] *UniPI at SemEval-2016 Task 4: Convolutional Neural Networks for Sentiment Classification*  
Giuseppe Attardi, Daniele Sartiano, Proceedings of SemEval-2016, Junio 16-17, 2016.  
<http://www.aclweb.org/anthology/S16-1033>
- [6] *A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks*  
Soujanya Poria, Erik Cambria, Devamanyu Hazarika, Prateek Vij  
<https://arxiv.org/pdf/1610.08815.pdf>
- [7] *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*  
Ye Zhang, Byron C. Wallace  
<https://arxiv.org/pdf/1510.03820.pdf>
- [8] *Procesamiento de Lenguaje Natural en Sistemas de Análisis de Sentimientos*  
Luciana Dubiau  
<http://materias.fi.uba.ar/7500/Dubiau.pdf>

- [9] *INESC ID at SemEval 2016 Task 4 A: Reducing the Problem of Out of Embedding Words*  
 Silvio Amir, Ramon F. Astudillo, Wang Ling, Mario J. Silva, Isabel Trancoso  
<http://www.ijke.org/vol1/20-R029.pdf>
- [10] *PotTS at SemEval 2016 Task 4: Sentiment Analysis of Twitter Using Character level Convolutional Neural Networks*  
 Uladzimir Sidarenka  
<http://www.aclweb.org/anthology/S16-1035>
- [11] *Minions at SemEval 2016 Task 4: or how to build a sentiment analyzer using off the shelf resources?*  
 Călin-Cristian Ciubotariu, Marius-Valentin Hrișca, Mihail Gliga, Diana Darabană, Diana Trandabăț and Adrian Iftene  
<http://www.aclweb.org/anthology/S16-1038>
- [12] *UofL at SemEval-2016 Task 4: Multi Domain word2vec for Twitter Sentiment Classification*  
 Omar Abdelwahab, Adel Elmaghrraby  
<http://www.aclweb.org/anthology/S16-1024>
- [13] *Construcción de un corpus marcado con emociones para el análisis de sentimientos en Twitter en español*  
 Camacho, V., Sidorov, G., Galicia, N.  
[http://www.cic.ipn.mx/~sidorov/corpus\\_marcado6emociones.txt](http://www.cic.ipn.mx/~sidorov/corpus_marcado6emociones.txt)
- [14] *Crowdsourcing for Beyond Polarity Sentiment Analysis: A Pure Emotion Lexicon*  
 Giannis Haralabopoulos, Elena Simperl.  
<https://arxiv.org/pdf/1710.04203.pdf>
- [15] *Emotions Evoked by Common Words and Phrases: Using Mechanical Turk to Create an Emotion Lexicon*  
 Saif M. Mohammad, Peter D. Turney  
<http://saifmohammad.com/WebDocs/Mohammad-Turney-NAACL10-EmotionWorkshop.pdf>
- [16] *Do You Feel What I Feel? Social Aspects of Emotions in Twitter Conversations*  
 Suin Kim, JinYeong Bak, Alice Oh  
<https://www.aaai.org/ocs/index.php/ICWSM/ICWSM12/paper/viewFile/4630/5041>

- [17] *#Emotional Tweets*  
 Saif M. Mohammad  
<https://arxiv.org/pdf/1510.03820.pdf>
- [18] *Corpus creado en: Construcción de un corpus marcado con emociones para el análisis de sentimientos en Twitter en español*  
 Camacho, V., Sidorov, G., Galicia, N.  
<https://arxiv.org/pdf/1610.08815.pdf>
- [19] *Lista de proyectos de programación distribuidas*  
 (Ultimo acceso: Marzo 2018)  
[https://en.wikipedia.org/wiki/List\\_of\\_distributed\\_computing\\_projects](https://en.wikipedia.org/wiki/List_of_distributed_computing_projects)
- [20] *MOSI: Multimodal Corpus of Sentiment Intensity and Subjectivity Analysis in Online Opinion Videos*  
 Amir Zadeh, Rown Zellers, Eli Pincus, Louis-Philippe Morency  
<https://arxiv.org/pdf/1606.06259.pdf>
- [21] *Convolutional Neural Networks for Sentence Classification*  
 Yoon Kim  
<https://arxiv.org/pdf/1408.5882.pdf>
- [22] *Creating English and Japanese Twitter Corpora for Emotion Analysis*  
 A. Danielewicz-Betz, H. Kaneda, M. Mozgovoy, and M. Purgina  
<http://www.ijke.org/vol1/20-R029.pdf>
- [23] *NLANGP at SemEval 2016 Task 5: Improving Aspect Based Sentiment Analysis using Neural Network Features*  
 Zhiqiang Toh, Jian Su  
<http://www.aclweb.org/anthology/S16-1045>
- [24] *Determining Word-Emotion Associations from Tweets by Multi-Label Classification*  
 Felipe Bravo-Marquez, Eibe Frank, Saif M. Mohammad, Bernhard Pfahringer  
<https://www.cs.waikato.ac.nz/~fbravoma/publications/wi2016a.pdf>
- [25] *An Overview of Multi-Task Learning in Deep Neural Networks*  
 Sebastian Ruder  
<https://arxiv.org/pdf/1706.05098.pdf>

- [26] *Twitter Sentiment in Data Streams with Perceptron*  
 Nathan Aston, Jacob Liddle, Wei Hu  
[http://file.scirp.org/pdf/JCC\\_2014021410235333.pdf](http://file.scirp.org/pdf/JCC_2014021410235333.pdf)
- [27] *UNIMELB at SemEval-2016 Tasks 4A and 4B: An Ensemble of Neural Networks and a Word2Vec Based Model for Sentiment Classification*  
 XingYi Xu, HuiZhi Liang, Timothy Baldwin  
<http://www.aclweb.org/anthology/S16-1027>
- [28] *Sentiment Analysis in Multiple Languages: Feature Selection for Opinion Classification in Web Forums*  
 Ahmed Abbasi, Hsinchun Chen, and Arab Salem  
[https://www.scss.tcd.ie/Khurshid.Ahmad/Research/Sentiments/K\\_Teams\\_Buchraest/2011\\_a12-abbasi.pdf](https://www.scss.tcd.ie/Khurshid.Ahmad/Research/Sentiments/K_Teams_Buchraest/2011_a12-abbasi.pdf)
- [29] *Lexicon-Based Methods for Sentiment Analysis*  
 Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, Manfred Stede  
[https://www.mitpressjournals.org/doi/pdfplus/10.1162/COLI\\_a\\_00049](https://www.mitpressjournals.org/doi/pdfplus/10.1162/COLI_a_00049)
- [30] *Compression and Machine Learning: A New Perspective on Feature Space Vectors*  
 D. Sculley, Carla E. Brodley  
<https://pdfs.semanticscholar.org/70e8/e1457aadbee439d47a2fe071007b1cf1dece.pdf>
- [31] *word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method*  
 Yoav Goldberg and Omer Levy  
<https://arxiv.org/pdf/1402.3722.pdf>
- [32] *Implementacion del esquema de red propuesto por Kim usando Python y Tensorflow*  
<https://github.com/dennybritz/cnn-text-classification-tf>
- [33] *Implementacion de un esquema de red capaz de soportar multitask usando Python y Tensorflow*  
<https://github.com/dhwajraj/deep-text-classifier-mtl>
- [34] *Understanding Convolutional Neural Networks for NLP*  
 Denny Britz. Noviembre 7 2015  
<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

- [35] *Implementing a CNN for Text Classification in TensorFlow*  
Denny Britz. Diciembre 11 2015  
<http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/>
- [36] *Multi-Task Learning in Tensorflow*  
Jonathan Godwin. Julio 2016  
<https://www.kdnuggets.com/2016/07/multi-task-learning-tensorflow-part-1.html>
- [37] *Serving a TensorFlow Model*  
[https://www.tensorflow.org/serving/serving\\_basic](https://www.tensorflow.org/serving/serving_basic)
- [38] *A Guide to TF Layers: Building a Convolutional Neural Network*  
<https://www.tensorflow.org/tutorials/layers>

## Herramientas Utilizadas

- [39] *Debian.* Distribucion de Linux.  
<https://www.debian.org/>
- [40] *Python 3.6.* Lenguaje de programación.  
<https://www.python.org/downloads/release/python-360/>
- [41] *PyCharm.* IDE para desarrollos basados en Python.  
<https://www.jetbrains.com/pycharm>
- [42] *Gensim.* Biblioteca de modelado de tópicos.  
<http://radimrehurek.com/gensim/>
- [43] *NumPy.* Paquete de Python para procesamiento numérico.  
<http://www.numpy.org/>
- [44] *SciPy.* Paquete de Python para computación científica.  
<http://www.scipy.org/>
- [45] *Cython.* Extensiones en lenguaje C para Python.  
<http://cython.org/>
- [46] *gRPC.* HTTP/2-based RPC framework  
<https://grpc.io/>
- [47] *SQLAlchemy.* Biblioteca de ORM para Python.  
<https://www.fusioncharts.com/>

- [48] *Flask*. Microframework para crear aplicaciones web.  
<http://flask.pocoo.org/>
- [49] *Jinja2*. Lenguaje de templating para Python.  
<http://jinja.pocoo.org/docs/2.10/>
- [50] *Mapbox*. Servicio para la creación de mapas online  
<https://www.mapbox.com>
- [51] *Bootstrap*. Toolkit para diseño de interfaces web.  
<http://getbootstrap.com/>
- [52] *FusionCharts*. Biblioteca de javascript para creación de graficos.  
<https://www.fusioncharts.com/>
- [53] *Gunicorn*. Servidor WSGI HTTP para UNIX.  
<http://gunicorn.org/>
- [54] *Pubnub*. Plataforma de infraestructura como servicio para la distribución de mensajes a nivel aplicación  
<https://www.pubnub.com/>
- [55] *TensorFlow*. Biblioteca opensource para Machine Learning  
<https://www.tensorflow.org/>
- [56] *RabbitMQ*. Open source message broker.  
<https://www.rabbitmq.com/>
- [57] *Docker*. Entorno de virtualizacion.  
<https://www.docker.com/>
- [58] *Heroku*. Plataforma de servicios en la nube para deploy de aplicaciones.  
<https://www.heroku.com/>
- [59] *ElephantSQL*. Servicio de bases de datos PostgreSQL en la nube.  
<https://www.elephantsql.com>
- [60] *Git*. Herramienta para el control de versiones.  
[http://git-scm.com/](http://git-scm.com)