Universidad de Granada

Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Sentiment Analysis

Práctica Final

Inteligencia de Negocio

Curso 2020 - 2021

José A. Martín Melguizo - (77561280J)

Correo: josemartin22@correo.ugr.es

Índice general

Capítulo 1

Descripción y análisis del problema

1.1. ¿Qué es el análisis de sentimientos?

El **análisis de sentimientos** consiste en el uso de técnicas de Procesamiento del Lenguaje Natural, analítica de textos y lingüística computacional para identificar y extraer información subjetiva u opinión expresada en texto.

Se utiliza en sistemas de recomendación, filtrado de mensajes/spam, en aplicaciones de negocios para predecir tendencias en función de las reseñas de los clientes o incluso seguir las opiniones sobre política, elaborar normas en función de la opinión de la gente (inteligencia de gobierno), etc...

El análisis de sentimientos se puede llevar a cabo a 3 niveles distintos en función de la granularidad, profundidad o detalle que se desee conseguir. Estos niveles son:

- **Análisis a nivel de documento:** en este nivel se analiza el sentimiento global del documento, clasificándolo como positivo, negativo o neutro (o usando cualquier otro sistema de calificación). En este nivel se asume que se expresa una valoración sobre un único ente, por lo que no sería aplicable para aquellos casos que hablen sobre varias entidades simultáneamente.
- **Análisis a nivel de oración:** en este nivel se divide el documento en oraciones individuales para extraer posteriormente la opinión que contiene cada una de ellas. La opinión de cada oración puede ser positivia, negativa o neutra (o tomar otro sistema de calificación al igual que el caso anterior).
- **Análisis a nivel de aspecto y entidad:** es el nivel más fino y con mayor detalle posible, una entidad está formada por distintos elementos o aspectos y sobre los que se expresa una opinión cuya polaridad puede ser distinta en cada caso. Este nivel es el que presenta un mayor desafío en la actualidad para los investigadores de dicho ámbito (sentiment analysis).

En nuestro caso nos **centraremos** en el primer nivel, la **clasificación** de la **polaridad** de una frase de opinión (en concreto de tweets), esto es, decidir si en una frase el autor muestra un sentimiento positivo, negativo o neutro.

1.2. Dataset para el desarrollo de la práctica

Se trabajará sobre el conjunto de datos en el dominio de Tweets del conjunto de datos `Sentiment140` que podemos descargalo de forma gratuita bien desde el enlace de Stanford o desde Google Drive, desde: http://help.sentiment140.com/for-students/.

Está formado por 1.600.000 tweets en el **conjunto de Training** etiquetados como polaridad positiva (clase '4') y polaridad negativa (clase '0').

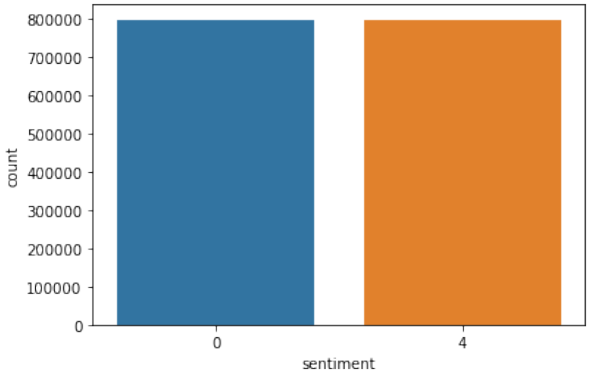

Figura 1.1: Distribución de las clases para datos de *training*

En el **conjunto de Test** tenemos 498 tweets etiquetados como polaridad positiva (clase '4'), polaridad neutra (clase '2') y polaridad negativa (clase '0').

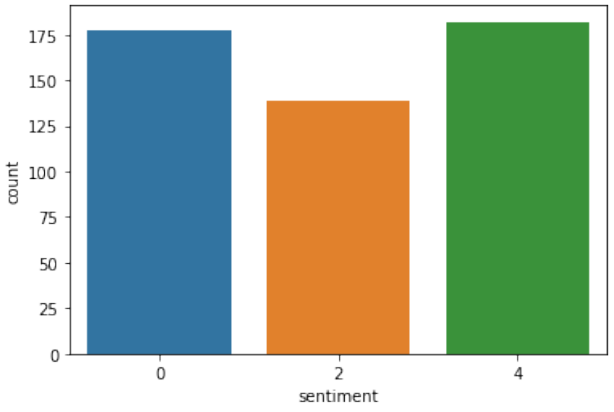

Figura 1.2: Distribución de las clases para datos de *test*

Más adelante comentaremos que para el caso de evaluar los **modelos preentrenados**, que nos permiten clasificar documentos con estas 3 polaridades (positiva, neutra y negativa), **mantendremos todas las clases** de instancias del conjunto de datos de test para comprobar su funcionamiento en totalidad.

Para el **resto de modelos** utilizados (en los que utilizaremos los datos de *training* para el entrenamiento de éstos), **descartaremos** las instancias etiquetadas con polaridad **neutra** (clase '2') del conjunto de *test*, ya que en entrenamiento no tenemos de éstas.

Capítulo 2

Descripción de los algoritmos

2.1. Métodos para la clasificación de documentos

La clasificación de un documento en función a la polaridad del sentimiento que expresa, puede llevarse a cabo de diversas formas. No hay un consenso (como en la mayoría de campos de la inteligencia artificial) sobre qué técnicas o algoritmos se deben usar para obtener mejores resultados en la clasificación de textos en unas situaciones u otras.

Aun así, podemos diferenciar los dos grandes grupos en los se encuentran dichas técnicas:

1. **Métodos supervisados:** Se basa en el empleo de algoritmos de aprendizaje automático de tipo supervisado, ya que se parte de un conjunto de textos previamente etiquetados (se conoce su polaridad) con los que se entrena un modelo que será usado posteriormente para clasificar un nuevo conjunto de textos (que en este ámbito es conocida como colección documental o *corpus*).

   Dichos algoritmos basan su funcionamiento en relaciones matemáticas creadas entre los elementos de entrenamiento. Éstos deben de ser representativos en cuanto a los elementos de test para el óptimo funcionamiento de éste (es el principal problema, no siempre se tiene un conjunto de entrenamiento lo suficientemente grande o bueno, a veces es difícil obtenerlo).

2. **Métodos no supervisados:** Resuelven el hecho de tener que contar con un conjunto de elementos preetiquetados (salva el problema de la dependencia del dominio). Estos métodos tratan de inferir la polaridad del sentimiento global de un documento a partir de la orientación semántica de las palabras o frases que lo conforman. Estos a su vez se dividen en:

   a) **Basados en diccionarios**: Hacen uso de un listado de palabras o frases junto a la polaridad que expresan, incluso con un grado de intensidad o fuerza de dicha polaridad. Tiene la desventaja de que a veces se pierde precisión ya que las palabras dependendiendo del contexto pueden tener una connotación más positiva o negativa, etc.

   Ese problema se resuelveçreando diccionarios concretos para un ámbito del lenguaje específico. Es decir el conjunto de palabras que lo forman se centra en un

dominio concreto, por ejemplo: medicina. Pero aún dentro de un mismo ámbito podría seguir ocurriendo ésto y habría que hilar más fino en la construcción de los mismos.

b) **Basados en relaciones lingüísticas**: Tratan de buscar ciertos patrones que expresen opiniones y sentimientos similares, con la mayor probabilidad posible, extrayendo las palabras que los formas para ser usadas posteriormente para clasificar textos mediante una función matematica que tenga en cuenta las palabras que lo contengan.

El enfoque no supervisado es elegante pero necesita apoyarse de alguna forma en datos que aporten una cierta orientación y sirvan de base de conocimiento. Por ello nos **centraremos** en métodos de **aprendizaje supervisado**.

2.2. Random Forest

Random Forest es una combinación de árboles de decisión o predictores tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de éstos. Supone por tanto una mejora sobre los árboles de decisión clásicos y la técnica de *bagging*.

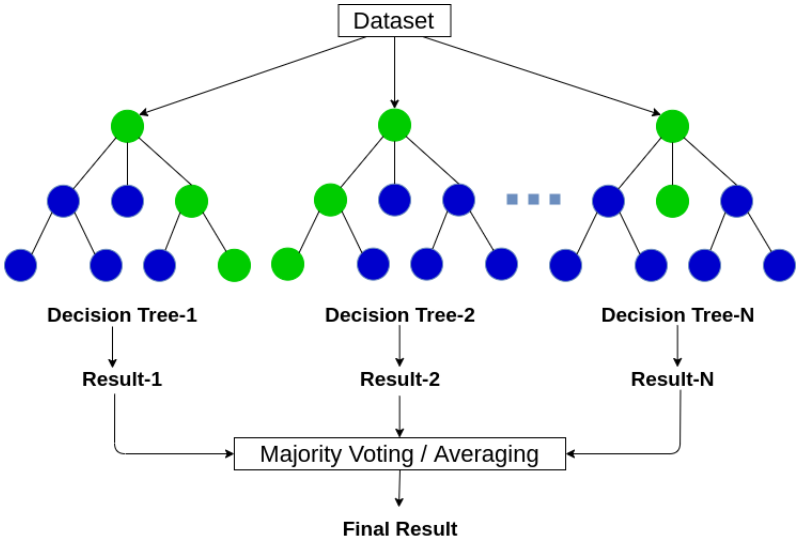

Figura 2.1: Representación de Random Forest

Este modelo permite tener múltiples árboles de decisión que son entrenados con una muestra de datos escogida con reemplazo a partir de la muestra original mediante *bootstrap*.

Además, cada árbol escoge un subconjunto de predictores que va a utilizar para intentar predecir, con el objetivo de evitar las correlaciones entre los distintos árboles, que era el principal problema de *bagging* (si existe un una variable predictora fuerte, esta era utilizado casi siempre en la primera decisión del árbol, haciendo por tanto que la mayoría de árboles se parecieran).

Además, es un modelo que ofrece unos resultados muy buenos en general a la hora de clasificar, ya que obtiene un bajo sesgo y varianza al tener múltiples árboles y promediar entre ellos (que además sabemos que no están correlacionados).

2.3. Máquinas de vectores de soporte

Las máquinas de vectores de soporte (del inglés, Support Vector Machine o SVM) son un grupo de algoritmos de aprendizaje supervisado en los que se trata de encontrar un hiperplano separador (denominado vector de soporte) que separe con la mayor distancia posible las distintas clases a las que pertenecen cada uno de los elementos.

De esta forma, el vector determina la frontera que sirve para clasificar un nuevo elemento.

Cuenta con una serie de parámetros que permitirán optimizar los resultados durante el proceso de clasificación, uno de ellos es el *kernel* y se utiliza cuando no es posible separar las muestras mediante una línea recta, plano o hiperplano de N dimensiones, permitiendo tal separación mediante otro tipo de funciones matemáticas como polinomios, funciones de base radial, etc.

Otro parámetro conocido es *regularization* o C, que permite una fluctuación de manera que se consideren ciertos errores en la clasificación y se evite el sobreaprendizaje en la manera de lo posible.

Y otro de ellos es *gamma* que determina la distancia máxima a partir de la cual una muestra pierde su influencia en la configuración del vector de soporte y *margin* que es la separación entre el vector y las muestras de cada clase más cercanas al mismo.

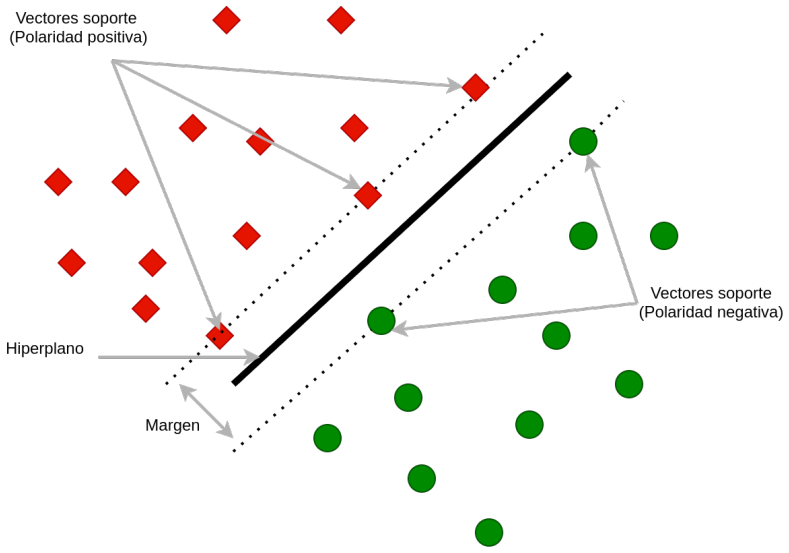

Figura 2.2: Representación de SVM para el problema

2.4. Naive Bayes

La familia de algoritmos Naive Bayes se basan en el conocido *Teorema de Bayes*:

Sea $\{A_1, A_2, \ldots, A_n\}$ *un conjunto de sucesos mútuamente excluyentes y exhaustivos, y tales que la probabilidad de cada uno de ellos es distinta de cero. Sea B un suceso cualquiera del que se conocen las probabilidades condicionales* $P(B|A_i)$.

Entonces la probabilidad de $P(A_i|B)$ viene dada por la expresión:

$$P(A_i|B) = \frac{P(B|A_i) * P(A_i)}{P(B)}$$

Para el caso de clasificación de textos, los sucesos son mútuamente excluyentes ya que no podemos considerar (en nuestro caso un tweet) que es de polaridad positiva y negativa al mismo tiempo y además son exhaustivos ya que esa calificación (positivo, negativo y neutro) son los únicos tipos que existen.

Además consideraremos que las palabras de un mismo mensaje no tienen ningún tipo de relación entre sí y por lo tanto es indiferente la posición que tienen dentro del texto al que pertenecen. (Esta condición es conocida en este ámbito como la suposición de bolsa de palabras o *bag of words*).

2.5. Métricas de evaluación

Para determinar el rendimiento de los resultados obtenidos por los distintos algoritmos, es necesario establecer unas métricas o medidas de evaluación que permitan evaluar de manera objetiva su eficacia a la hora de clasificar los ejemplos que se le proporiconen a éstos.

Para ello, es importante tener en cuenta los cuatro posibles estados de un ejemplo clasificado, por ejemplo con respecto a una clase C:

1. **Verdaderos positivos** (True Positives o TP): son aquellos que han sido marcados de manera correcta como pertenecientes a la clase C.

2. **Falsos positivos** (False Positives o FP): son aquellos marcados como de clase C, pero que en realidad no pertenecen a dicha clase, es decir, han sido clasificados de forma incorrecta.

3. **Verdaderos negativos** (True Negatives o TN): son aquellos que no son de la clase C y han sido clasificados correctamente.

4. **Falsos negativos** (False Negatives o FN): son aquellos clasificados como no pertenecientes a la clase C, pero en realidad sí lo son y por tanto no se han clasificado correctamente.

Teniendo en cuenta los cuatro posibles estados anteriores, podemos definir las siguientes medidas que serán usadas para evaluar el rendimiento de nuestros modelos:

1. **Exactitud o *Accuracy***: Es la medida de rendimiento más simple e intuitiva. Representa la relación entre las predicciones correctas sobre el total de las predicciones realizadas.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

   Podemos pensar que un modelo con mayor exactitud que otro es mejor. Eso sería así en el caso de que el número de elementos de cada clase es el mismo, es decir, el corpus está balanceado. En caso contrario, es necesario hacer uso de otro tipo de medidas como la precisión o la exhaustividad (valoran el rendimiento sobre clases individuales).

2. **Precisión**: Es la razón entre el número de documentos clasificados correctamente como pertenecientes a la clase C y el número total de documentos que han sido clasificados por el modelo como de clase C. Mide la proporción de identificaciones positivas que son realmente correctas.

$$Precision = \frac{TP}{TP + FP}$$

3. **Exhaustividad o *Recall***: Es la razón entre los documentos clasificados correctamente como pertenecientes a la clase C y la suma de todos los documentos de la clase C. Se puede entender como la capacidad que tiene el modelo de construir de manera correcta las clases.

$$Recall = \frac{TP}{TP + FN}$$

4. **Media armónica ponderada o F-score**: Puede ser interpretado como un promedio ponderado de la *precisión* y el *recall* a través de un parámetro β que permite otorgar una mayor importancia a una que otra, donde el rango de valores oscila entre 0 y 1.

$$F1 = (1 + \beta^2) * \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

   Se ayudará a penalizar los valores muy bajos en una u otra medida. Es frecuente que la precisión y el recall tengan el mismo peso en la fórmula, es decir, con un valor $\beta = 1$. A esta configuración se le conoce como **F1-score** y es el que **utilizaremos**.

Es de la forma:

$$\frac{2 * Precision * Recall}{Precision + Recall}$$

Para el caso en el que se cuenten con **más de 2 clases** (solo lo tendremos en cuenta al utilizar modelos preentrenados, ya que predeciremos tanto polaridad negativa, neutra y positiva. En nuestro conjunto de entrenamiento solo tenemos polaridad positiva y negativa), se debe **calcular** cada una de las **métricas anteriores** para **cada clase** y combinarlas entre ellas para obtener una **medida global**.

La función `classification_report` de *sklearn* nos proporciona:

1. **Macro-averaging**: Se calculan las medias para cada una de las clases, tanto de precisión, recall y f1-score. Esta medida no tiene en cuenta el posible desbalanceo entre el número de ejemplos de cada clase. Por ejemplo para el caso de la precisión sería:

$$Macro - precision = \frac{\sum_{i=1}^{n} Precision_i}{n}$$

2. **Weighted-averaging**: Resuelve el problema de tener un *corpus* desbalanceado y con más de dos clases. Se pondera cada componente de la fórmula en base al peso que cada clase tiene dentro del sistema. De nuevo, para el caso de la precisión sería:

$$Weighted - precision = \frac{\sum_{i=1}^{n} Precision_i * P_i}{n}$$

Comentar que los algoritmos anteriormente nombrados serán entrenados mediante **validación cruzada** o *cross-validation*, de forma que se reducirá la dependencia entre la partición de los datos usada para la fase de entrenamiento y test.

Se dividirá el conjunto de documentos o *corpus* en 5 particiones de iugal tamaño, de forma que que 4 serán para entrenar el modelo y el sobrante para la evaluación. Esto se repetirá 5 veces de forma que al final se promediará el valor de las métricas para tener una medida más robusta.

Capítulo 3

Estudio experimental

Para abordar el problema, se nos han propuesto una serie de aproximaciones en orden creciente de complejidad:

- Comenzaremos utilizando modelos de análisis de sentimientos ya preentrenados.
- A continuación construiremos nuestro propio método de análisis de sentimientos (crearemos un clasificador haciendo uso de los algoritmos explicados anteriormente y aplicando un preprocesado a los datos para mejorar los resultados).
- Y por último (como ampliación de la práctica) utilizaremos modelos mucho más avanzados.

3.1. Empleo de métodos de análisis de sentimientos preentrenados (SAMs)

Estos métodos ya han sido preentrenados con un conjunto de textos que representan de forma general el tipo de polaridad (positiva, negativa, neutra) que puede expresar una oración o documento.

No será necesario entrenar con nuestros datos, sólamente tendremos que utilizar los datos de test para predecir directamente la polaridad que expresan nuestros tweets.

3.1.1. VaderSentiment

VADER *(Valence Aware Dictionary and sEntiment Reasoner)* es una herramiento de análisis de sentimientos basada en reglas y léxico que está específicamente en sintonía con los sentimientos expresados en las redes sociales. Es completamente de código abierto y se instala como una librería más para python a través de *pip*.

La puntuación *compound* se calcula sumando las puntuaciones de valencia de cada palabra en el léxico, se ajusta de acuerdo con las reglas y luego se normaliza para estar entre -1 (extremo más negativo) y +1 (extremo más positivo).

Esta es la métrica más útil si desea una única medida unidimensional de sentimiento para una oración determinada.

El **significado** de dicha **puntuación** es el siguiente:

- sentimiento positivo : $compound >= 0,05$
- sentimiento neutral : $compound > -0,05$ y $compound < 0,05$
- sentimiento negativo : $compound <= -0,05$

Los **resultados obtenidos** para el conjunto de *test* han sido los siguientes:

Polaridad	Precision	Recall	F1-score
Negativo	0.84	0.64	0.73
Neutro	0.67	0.70	0.68
Positivo	0.68	0.81	0.74
Macro avg	0.73	0.72	0.72
Weighted avg	0.73	0.72	0.72
Accuracy			0.72

Cuadro 3.1: Resultados para VaderSentiment

3.1.2. TextBlob

TextBlob es una biblioteca de Python (2 y 3) para procesar datos textuales. Proporciona una API simple para sumergirse en tareas comunes de procesamiento del lenguaje natural (NLP), como etiquetado de parte del discurso, extracción de frases nominales, análisis de sentimientos, clasificación, traducción y más.

Los **resultados obtenidos** para el conjunto de *test* han sido los siguientes:

Polaridad	Precision	Recall	F1-score
Negativo	0.81	0.48	0.60
Neutro	0.59	0.70	0.64
Positivo	0.63	0.79	0.70
Macro avg	0.68	0.66	0.65
Weighted avg	0.68	0.65	0.65
Accuracy			0.65

Cuadro 3.2: Resultados para TextBlob

A la vista de los resultados, pese a no haber visto una sola muestra de entrenamiento, los resultados son aceptables, sobretodo en el caso de VaderSentiment (la cual está entrenada con textos de redes sociales) y es más afín a nuestro problema.

Comentar que TextBlob también nos permitía a través de su API entrenar un modelo de Naive Bayes (ahora sí utilizando nuestros datos de entrenamiento) con el que hemos obtenido un precisión del 72 % aprox. Los resultados serán comentados más en profundidad en el análisis que haremos más adelante.

3.2. Creación de un propio método de análisis de sentimientos

La construcción de un clasificador de documentos de texto basado en un sistema de aprendizaje automático consta de varias etapas. En primer lugar es necesario preparar los datos del *corpus* antes de pasárselos a los algoritmos. Se deben limpiar y normalizar de forma que se reduzca o eliminen datos innecesarios para el aprendizaje o que puedan influir de manera negativa en éste.

A continuación cada uno de los tweets se someterá a un proceso de **tokenización**, el cual dividiremos el texto en unidades de significado conocidas como *tokens* (suelen ser palabras).

A partir de ellos se extraerán las características que representen a los mensajes y se aplicarán métodos para reducir su número, como podría ser la aunación de términos morfológicamente similares para reducir su número. Finalmente se ponderarán las características en función de la importancia que se les quiera dar y con ellas se entrenarán los clasificadores.

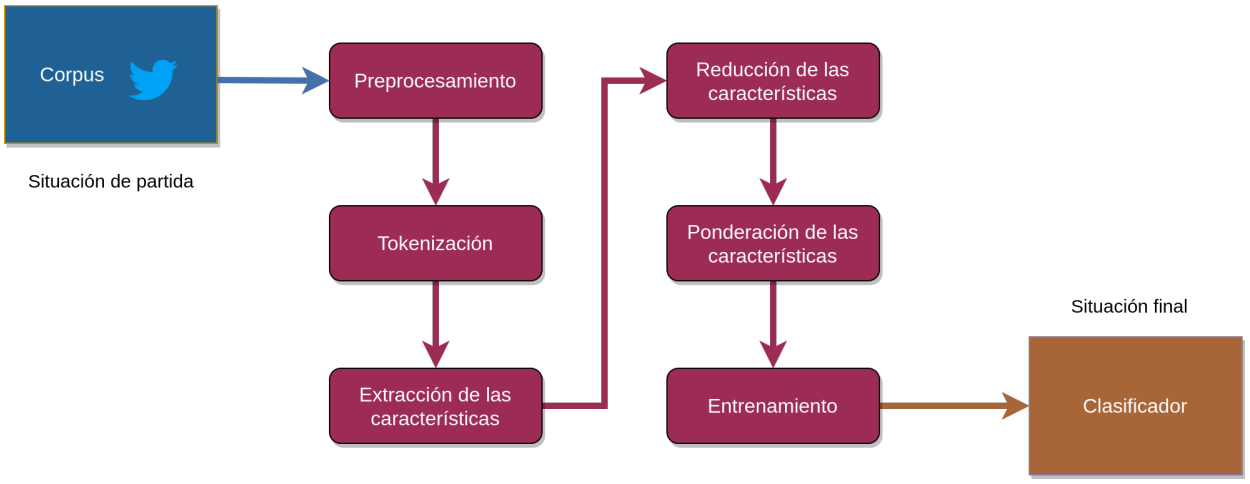

Figura 3.1: Fases para la creación del clasificador

3.2.1. Preprocesamiento

Partimos de una colección de tweets en los que tenemos todo tipo de información: urls, menciones, hashtags, repeticiones de caracteres, mezcla de letras mayúsculas y minúsculas etc. El objetivo de esta fase es tratar de limpiar y normalizar los datos que puedan influir de manera negativa a la hora de clasificar.

A continuación comentamos una serie de tratamientos que hemos aplicado a cada uno de los tweets de forma que se limpien esos datos pero sin provocar una pérdida excesiva de la polaridad de los mismos:

1. **Decodificando HTML**: Parece que la codificación HTML no se ha convertido a texto y en algunos casos se encuentran terminaciones como '&amp' y '&quot'. Por lo que

haremos uso de la librería **BeautifulSoup** para extraer preprocesar esos tweets que contengan información de este tipo.

2. **Eliminando @mention**: La segunda parte del preprocesamiento será eliminar las menciones que unos usuarios hacen a otros. Esta información no aportará nada de valor a la hora de construir un modelo de análisis de sentimiento. Utilizaremos una expresión regular.

3. **Eliminando URL links**: Sucede de forma similar que con las menciones, es información relevante en el tweet pero para fines de análisis de sentimientos, podemos ignorarlos. Utilizaremos una expresión regular.

4. **Hashtags y números**: A veces, el texto utilizado con un hashtag puede proporcionar información útil sobre el tweet. Puede ser un poco arriesgado deshacerse de todo el texto junto con el hashtag. Simplemente eliminaremos el '#'. Haré esto en el proceso de limpieza de todos los caracteres que no son letras, incluidos los números.

5. **Normalización a minúsculas**: Una misma palabra escrita de forma completa en mayúsculas y en minúsculas es tratada de forma distinta por un algoritmo de aprendizaje automático. Para evitar que esto suceda, convertiremos todos los mensajes a su equivalente en letras minúsculas, de la misma forma que haría un *LowerCaseFilter*.

De forma que la función para la limpieza del tweet nos queda de la forma:

```
from bs4 import BeautifulSoup
import re

pat1 = r'@[A-Za-z0-9]+'
pat2 = r'https?://[A-Za-z0-9./]+'
combined_pat = r'|'.join((pat1, pat2))

def tweet_cleaner(text):

    # HTML Decoding
    soup = BeautifulSoup(text, 'lxml')

    # @Menciones, URL links
    souped = soup.get_text()
    stripped = re.sub(combined_pat, '', souped)
    try:
        clean = stripped.decode("utf-8-sig").replace(u"\ufffd", "?")
    except:
        clean = stripped

    # Hashtag, numeros
    letters_only = re.sub("[^a-zA-Z]", " ", clean)

    # LoweCaseFilter
    lower_case = letters_only.lower()

    words = tok.tokenize(lower_case)

    return (" ".join(words)).strip()
```

3.2.2. Tokenización

Una vez completado el proceso de normalización de los mensajes, la siguiente etapa es la de tokenización. Consiste en dividir el texto en unidades más pequeñas llamadas tokens las cuales tienen un significado propio y que normalmente suelen ser cada una de las palabras del mensaje. Por lo tanto **separaremos** los **términos** del tweet por **espacios en blanco**.

3.2.3. Extracción de las características

A partir de los tokens obtenidos en el paso anterior, se define una manera de representar los mensajes, creando así las características de cada tweet. Lo habitual en clasificación de documentos es hacer uso del modelo de bolsa de palabras ya mencionario anteriormente (también conocido como del inglés como *bag of words*) donde cada mensaje se representa mediante sus tokens **sin tener en cuenta** el **orden** en los que éstos aparecen. En nuestro caso **cada característica** será un *unigrama* o token individual.

3.2.4. Reducción de las características

Esta fase suele ser opcional y el objetivo es reducir la dimensionalidad o el número de características del *corpus* mediante la la **eliminación** de algunos de ellos, **aunación** de tokens morfológicamente similares, la **conversión** de tokens parecidos a otro token más general.

Existen técnicas para llevar a cabo cada una de esas tareas comentadas:

1. **Eliminación de palabras vacías**: En todos los idiomas hay un conjunto de palabras pequeño que suele tener una frecuencia muy alta y que por norma general sirven para construir las oraciones pero que carecen de información. Estas palabras son las preposiciones, conjunciones, artículos, etc.

   Se define una lista con dichas palabras, conocidas como *stopwords* y serán eliminadas de cada uno de los tweets, de forma que se reducirá el número de características del mismo.

2. **Stemming**: Es un método de normalización lingüística que transforma cada palabra a su raíz por medio de la supresión de prefijos, sufijos e infijos. Por ejemplo las palabras: perro, perrito, perritas se reduciría a una misma raiz perr.

3. **Lematización**: Es otro método de normalización lingüística en el que cada palabra se transforma en su lema mediante el uso de un diccionario y un proceso de análisis morfológico.

   Por ejemplo podría considerarse el cambio de género o número a una misma forma. Se trata de reducir la variabilidad de una palabra. Es más costoso que el stemming, pero como resultado tenemos un reducción más fina de los términos.

De forma que la función para la reducción del tweet nos queda de la forma:

```python
# English Stopwords
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def preprocesar_texto_tweet(tweet):

    # Ya los teniamos convertidos a minuscula

    # Eliminamos las palabras vacias (Filtramos aquellas que no lo son)
    tweet_tokens = word_tokenize(tweet)
    filtered_words = [word for word in tweet_tokens if word not in
   stop_words]

    # Stemming
    ps = PorterStemmer()
    stemmed_words = [ps.stem(w) for w in filtered_words]

    # Lematizacion
    lemmatizer = WordNetLemmatizer()
    lemma_words = [lemmatizer.lemmatize(w, pos='a') for w in stemmed_words
   ]

    return " ".join(lemma_words)
```

A continuación mostramos el top de los 50 términos más frecuentes utilizados en tweets con polaridad negativa sin considerar la reducción de características y tras considerarla:

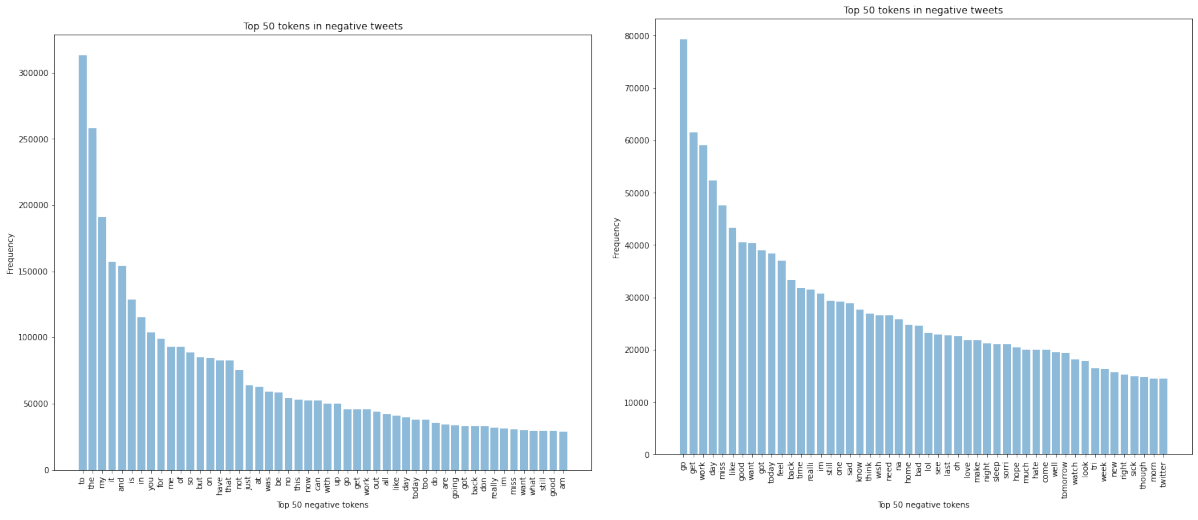

Figura 3.2: Top 50 términos negativos antes y después de preprocesar

Podemos observar que antes aparecían palabras como: *to, the, my, it, and* que son **palabras** carentes de significado o **vacías** (stopwords) y no reflejan el sentimiento negativo.

Tras preprocesar podemos ver como entre las palabras más frecuentes aparecen: *get, work, miss, feel, sad, wish, bad* las cuales son mucho más **representativas** para expresar la **polaridad negativa**.

De la misma forma ocurre con los 50 términos más frecuentes utilizados en tweets que con polaridad positiva. De la misma forma que antes, las gráficas resultantes son:

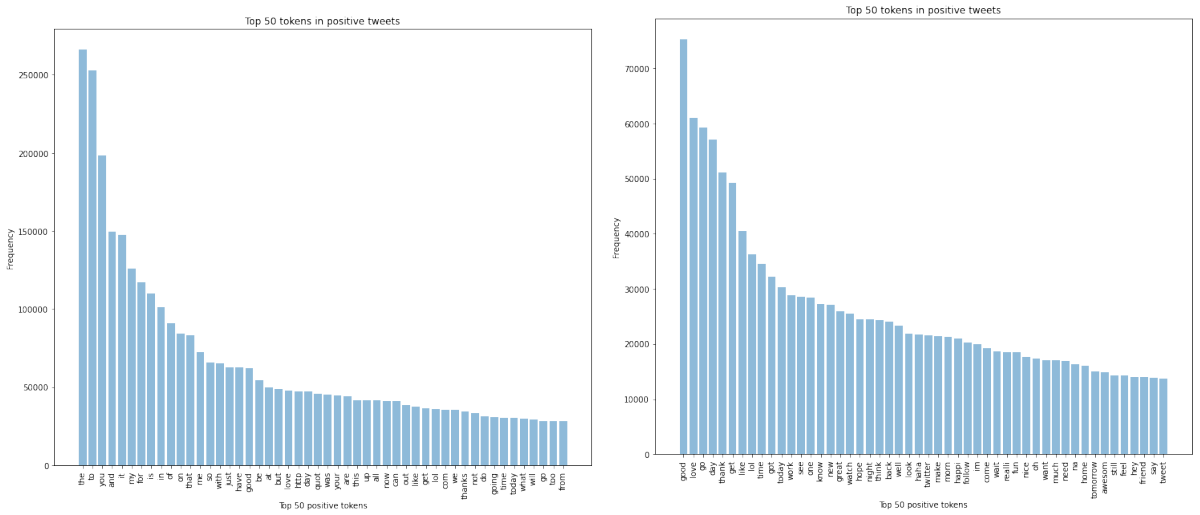

Figura 3.3: Top 50 términos positivos antes y después de preprocesar

Podemos observar que vuelven a aparecer de hecho las mismas palabras como: *the, to, you, it, and* que son **palabras vacías** y además de no reflejar el sentimiento positivo, coinciden con las negativas y por lo tanto pueden causar confusión para el clasificador.

Tras preprocesar podemos ver como entre las palabras más frecuentes aparecen: *good, love, thank, like, lol, great* las cuales son mucho más **representativas** para expresar la **polaridad positiva**.

La **reducción de la dimensionalidad** es bastante notable. Como ya comentaremos más adelante, utilizaremos un total de 6000 instancias para el entrenar los clasificadores.
Tras obtener el esquema tf-idf para cada uno de los documentos, obtenemos una matriz de: 6000 rows $\times$ 12554 columns.

Una vez preprocesado todos los datos, obtenemos una matriz de: 6000 rows $\times$ 7649 columns, lo que supone una reducción de dimensionalidad de un **39 %** y por lo tanto los **tiempos de ejecución** y los **resultados** como veremos más adelante, **mejorarán** significativamente.

3.2.5. Ponderación de las características

Las características extraídas pueden ser consideradas con la misma importancia o podemos asignarle distintos pesos en función de algún tipo de criterio, como por ejemplo la frecuencia de aparición de éstas en el propio tweet o en relación con la totalidad e los tweets.

Existen distintos métodos de ponderación, los cuales se han visto con más profundidad en la asignatura de Recuperación de Información, pero aquí voy a mostrar directamente el más famoso y utilizado en el ámbito de *Text mining* e *Information retrieval*, que es el esquema de **ponderación TF-IDF**.

Definimos a continuación los dos conceptos que componen a dicho esquema de ponderación:

- **Term Frequency** o TF:

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

  La frecuencia del término i en el documento j se define como el número de veces que aparece una palabra en un documento dividido por el número total de palabras del documento.

- **Inverse Document Frequency** o IDF:

$$idf_i = log\left(\frac{N}{df_t}\right)$$

  La frecuencia documental inversa de un término i se define como el logaritmo del número de documentos del *corpus* dividido entre el número de documentos en los que aparece el término i.

  Favorece por tanto a las palabras que aparecen en pocos documentos y penaliza a aquellas que suelen aparecer en la mayoría (son las que no nos ayudan a discriminar los documentos).

Finalmente, el **esquema TF-IDF**, que será la ponderación de un término i en un documento j, se define como el producto de ambos conceptos:

$$w_{ij} = tf_{ij} \times log\left(\frac{N}{df_t}\right)$$

No será necesario calcular éstos datos de forma manual, podemos hacer uso de la clase `TfidfVectorizer` proporcionada por *sklearn*.

3.3. Comparación y análisis de los modelos empleados

Al haber 1.6 millones de tweets, los tiempos de ejecución pueden dispararse, por lo que emplearemos, como se nos ha recomendado, un subconjunto representativo y balanceado del conjunto de Train.

En concreto, vamos a utilizar una pequeña submuestra de 6000 instancias (3000 para cada una de las polaridades, positiva y negativa). He intendado hacerlo con 4000 para cada una de ellas, pero la memoria RAM se llena muy rápido y el ordenador se quedaba colgado hasta tener que apagarlo forzósamente.

A continuación, mostramos las tablas con los resultados de las métricas de evaluación obtenidas para cada uno de los modelos considerados (utilizando y sin utilizar el preprocesamiento anteriormente comentado).

3.3.1. Tablas con los resultados

	Sin preprocesamiento			Con preprocesamiento		
Polaridad	Precision	Recall	F1-score	Precision	Recall	F1-score
Negativo	0.68	0.67	0.67	0.79	0.68	0.70
Positivo	0.68	0.70	0.69	0.69	0.80	0.77
Macro avg	0.68	0.68	0.68	0.74	0.74	0.73
Weighted avg	0.68	0.68	0.68	0.74	0.74	0.73
Accuracy			0.68			0.74

Cuadro 3.3: Resultados para Random Forest

	Sin preprocesamiento			Con preprocesamiento		
Polaridad	Precision	Recall	F1-score	Precision	Recall	F1-score
Negativo	0.73	0.68	0.70	0.80	0.63	0.70
Positivo	0.71	0.75	0.73	0.70	0.85	0.77
Macro avg	0.72	0.72	0.72	0.75	0.75	0.75
Weighted avg	0.72	0.72	0.72	0.75	0.75	0.75
Accuracy			0.72			0.75

Cuadro 3.4: Resultados para SVM

	Sin preprocesamiento			Con preprocesamiento		
Polaridad	Precision	Recall	F1-score	Precision	Recall	F1-score
Negativo	0.73	0.74	0.73	0.77	0.70	0.73
Positivo	0.74	0.73	0.74	0.73	0.80	0.76
Macro avg	0.74	0.74	0.74	0.74	0.75	0.74
Weighted avg	0.74	0.74	0.74	0.74	0.75	0.74
Accuracy			0.74			0.74

Cuadro 3.5: Resultados para Naive Bayes

A la vista de los resultados obtenidos, lo primero que nos llama la atención es que en **todos** los **algoritmos**, en la mayoría de métricas, los **resultados mejoran** cuando aplicamos el **preprocesamiento** a los documentos de partida. Las diferencias más grandes en cuanto a *accuracy* nos referimos, se pueden observar sobretodo en Random Forest y también en SVM.

Naive Bayes parece ser igual de robusto indiferentemente si aplicamos o no el preprocesados a los datos. Sin embargo, si nos fijamos el preprocesamiento ayuda sobretodo a **mejorar la clasificación** de los casos **positivos**.

Esto ocurre en todos los casos, se obtiene un *recall* a la hora de clasificar los tweets con **polaridad positiva** un 15 % superior tras preprocesar en Random Forest, un 10 % superior en SVM y un 7 % superior en Naive Bayes.

A diferencia del *recall* para los tweets con **polaridad negativa** que baja un 5 % en todos los algoritmos aproximadamente. Esto puede ocurrir bien porque el subconjunto de entrenamiento de nuestro dataset es bastante pequeño y no es demasiado representativos para los casos negativos de aquellos términos que expresan la polaridad negativa.

Si comparamos los resultados de *accuracy* obtenidos en nuestros clasificadores propios contra los modelos preentrenados (VaderSentiment y TextBlob) que comentamos anteriormente, podemos observar que **nuestros modelos** ofrecen **mejor comportamiento** obtniendo unos resultados de media un 10 % superiores respecto a TextBlob y un 3 % superiores a VaderSentiment.

A continuación mostramos en una tabla todos los clasificadores/modelos empleados junto a su accuracy

Clasificador	*Accuracy*
VaderSentiment	0.72
TextBlob	0.65
Random Forest	0.74
Support Vector Machine	0.75
Naive Bayes	0.74

Comentar que VaderSentiment, pese a no haber sido entrenado con nuestros datos de tweets, obtiene buenos resultados ya que este clasificador está en sintonía con los sentimientos expresados en las redes sociales, por lo tanto la forma de expresión de los usuarios de Tweeter le es familiar.

En cuanto a propios clasificadores, los mejores resultados los hemos obtenido con SVM (en concreto con el *kernel* de base radial o *rbf*) y tras hacer una **búsqueda** de los **mejores parámetros** con `GridSearchCV`

Obtenemos aprox. un 76 % considerando el modelo SVC(C=200, class_weight='balanced', gamma=0.001), siendo C el factor de regularización inversamente proporcional a éste y un valor gamma (para el coeficiente del *kernel*) de 0.001, además de una ponderación balanceada en el pesos de las clases para la etapa de entrenamiento del clasificador.

3.4. Modelos más avanzados

A continuación, como ampliación de la práctica, **comentaremos** a grandes rasgos la **teoría** subyacente por debajo de estos modelos y formas de representación del texto que son más complejas pero con mejor comportamiento en general. Además **incluyo en la entrega el código** en *jupyter-notebook* de los experimentos realizados que comentaré más adelante.

3.4.1. Word2vec

Es una herramienta que proporciona una implementación eficiente de las arquitecturas continuas de bolsa de palabras o *bag of words* y omisión de la gramática para computar **representaciones vectoriales** de palabras que logran capturar el **significado semántico** de las mismas (a diferencia del esquema de representación tf-idf que consideran las palabras por separado como características).

Word2vec toma un *corpus* de texto como entrada y produce los vectores de palabra como salida. Primero se **construye un vocabulario** a partir de los datos del texto de entrenamiento y luego se **aprende la representación vectorial**. de las palabras. Las palabras resultantes pueden utilizarse como características para entrenar nuestros modelos de aprendizaje automático.

Para entender cómo funciona la representación vectorial de una palabra, y poder encontrar palabras cercanas o similares a otras, se utiliza como concepto de **distancia** la **similitud coseno**. Por ejemplo si se ingresa la palabra 'España', las palabras similares a ésta estarán cerca de ella, como podrían ser 'Francia', 'Alemania' etc.

De la misma forma ocurriría con adjetivos. En la siguiente imagen se muestra como **aquellos** que **son positivos** están **más cerca entre sí**, de igual forma ocurre con los negativos.

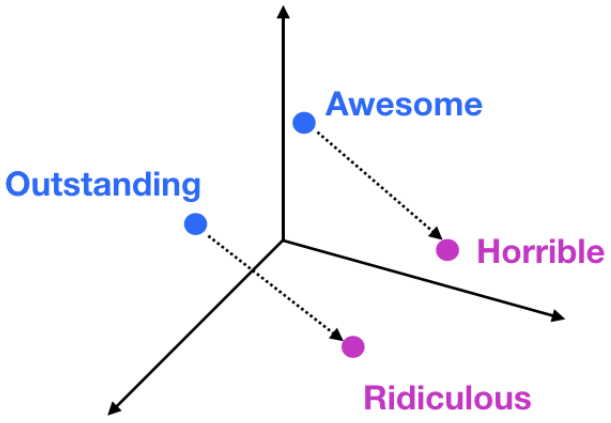

Figura 3.4: Representación word2vec

Para entrenar este modelo, se toman en consideración las palabras que rodean esa palabra de tamaño de ventana particular. Hay diferentes formas de derivar los vectores de

incrustación de palabras. Word2vec es uno de esos métodos donde se utiliza el modelo de incrustaciones neuronales para aprender eso. Utiliza las siguientes **dos arquitecturas** para lograr esto.

- **Bolsa Contínua de Palabras o CBOW**: del inglés Continuous Bag of Words, se trata de un modelo que predice la palabra en consideración con las palabras dadas como contexto dentro de una ventana específica. La capa oculta tiene un número de dimensiones igual a las necesarias para representar la palabra actual en la capa de salida.

  En el siguiente ejemplo se muestra una ventana de tamaño 2 para predecir el vector de la palabra 'awesome' dada la oración: 'Restaurant was awesome this time'

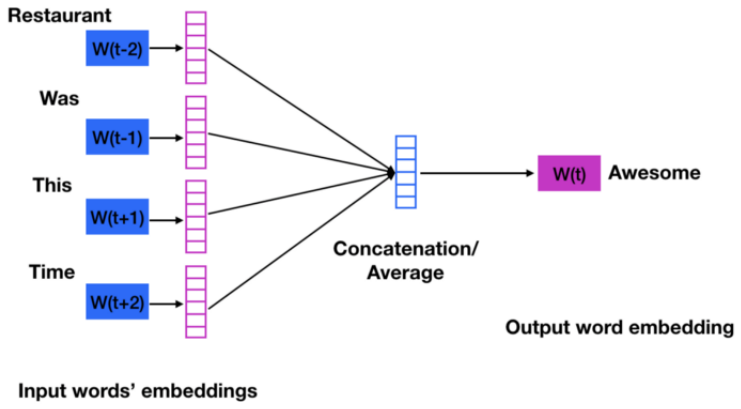

Figura 3.5: Arquitectura CBOW

- **Skip Gram**: Es opuesto a CBOW, se predice las palabras más similares o alineadas para las palabras de contexto circundantes en la ventana específica dada una palabra actual. A continuación se muestra el ejemplo anterior con un tamaño de ventana 2 para esta arquitectura:

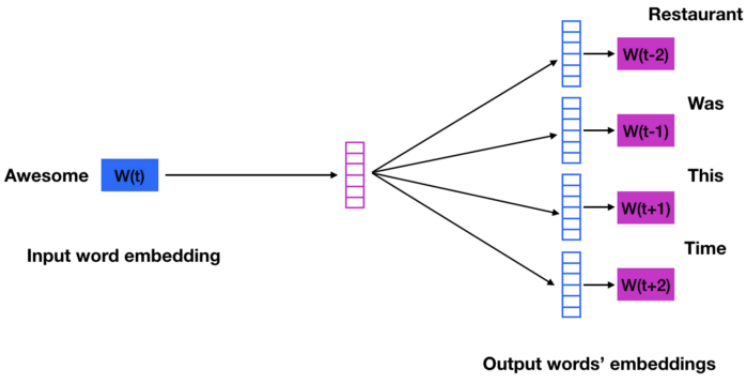

Figura 3.6: Arquitectura Skip Gram

La capa de entrada contiene la palabra actual y la capa de salida contiene las palabras de contexto. La capa oculta contiene el número de dimensiones en las que queremos representar la palabra actual presente en la capa de entrada.

Centrándonos un poco más en nuestro problema a resolver:

Los **vectores de palabras** también se pueden utilizar para **derivar clases de palabras** a partir de grandes conjuntos de datos. Esto se logra realizando **agrupaciones** de **K-medias** encima de los vectores de palabras.

Podría resultarnos útil si se intenta agrupar todos los tweets con la misma polaridad para solventar de esta forma el problema de clasificación en función de la polaridad del documento.

3.4.2. Modelos de Deep Learning

Una buena aproximación que obtiene un *accuracy* de 0.79 con una red neuronal de tipo LSTM es la que se encuentra en el siguiente enlace:
https://www.kaggle.com/abhineethmishra/twitter-sentiment-analysis
Utiliza junto con este tipo de redes el modelo *word2vec* y además entrena con toda la colección completa.

No he podido realizar pruebas con este tipo de hibridación ya que los tiempos de cómputo se disparaban. Sin embargo, para probar un poco *word2vec* he realizado un pequeño experimento junto a una simple red de tipo FFNN con 3 capas, pero el *accuracy* obtenido apenas superaba el 0.35.

Otra **aproximación** que he realizado está basada en el código:
https://www.kaggle.com/oksanakalytenko/glove-lstm/notebook
en el que se obtiene un *accuracy* de 0.77 entrenando con toda la colección completa.
Utiliza de forma similar a la anterior, redes neuronales **LSTM** pero **junto a Glove**, que es un *word embedding* desarrollado por la Universidad de Standford.

En mi caso he obtenido un *accuracy* de **0.71** entrenando con un subconjunto de entrenamiento del 3.75 % total (que son 60.000 instancias). Pese a utilizar tan pocos datos, ya tardó unos 42 minutos aprox. en ejecutar. La gráficas de aprendizaje obtenidas son:

Podemos observar que pese al parecer estar presente el sobreaprendizaje al inicio y en la sexta época de aprendizaje, al final ese incremento de rendimiento en training se ve reflejado en validación y no hay demasiada varianza o distancia entre ambas lineas, por lo que al final obtenemos un buen rendimiento del modelo entrenado.

Las redes LSTM al poder **retroalimentarse** pueden reaprender más rápidamente por ello obtenemos mejores resultados que con las FFNN del experimento anterior.

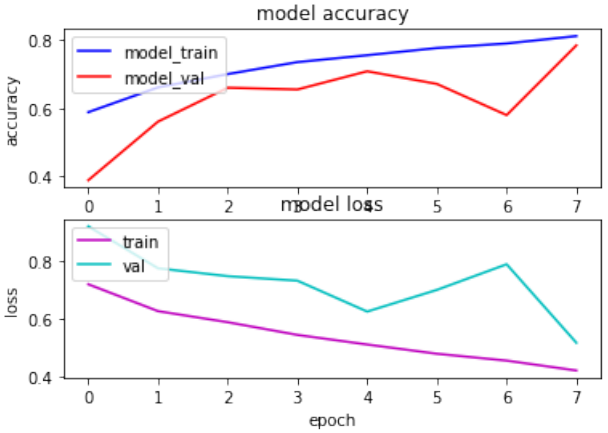

Figura 3.7: Gráficas de aprendizaje

Capítulo 4

Planteamiento de futuro

4.1. Problemas que presenta el análisis de sentimientos

Algunos de los problemas que enunciaremos a continuación están directamente relacionados o podríamos decir que son problemas que se heredan del Procesamiento del Lenguaje Natural, entre ellos:

- El sentimiento de las palabras a menudo depende del **contexto en el que estén ubicadas**, por ejemplo no tendría el mismo significado el verbo "morir" cuando nos referimos al fallecimiento de una persona (sentimientos negativos), que cuando utilizamos "morir de la risa" para denotar una situación graciosa (sentimientos positivos).
- La **ambigüedad** es uno de los elementos lingüísticos más sutiles y complicados de resolver dentro del PLN. Las personas somos capaces de deducirlos dentro de un contexto y en base a nuestras experiencias, pero los ordenadores no cuentan con demasiados recursos para ello. Esto se complica cuando el nivel de granuralidad cada vez es más fino (análisis a nivel de aspecto y entidad).
- La **ironía** de las expresiones utilizadas por las personas o cualquier otra **expresión coloquial** que no se suele utilizar por mucha gente en general.
- Detección de las **frases negadas**, ya que palabras como "no" pueden invertir la polaridad del resto de palabras a las que acompañan (y no siempre la negación invierte el sentimiendo de éstas).

  Por ejemplo: "La comida está buena", si añadimos el "no" delante del verbo cambiaría totalmente la polaridad de ésta.

  Sin embargo, en la oración "No cabe duda de que la comida está buenísima"', sigue teniendo la polaridad positiva.

4.2. ¿Qué harías si puedes trabajar durante 6 meses en el problema?

Bibliografía

Preprocesado

https://github.com/tthustla/twitter_sentiment_analysis_part1/blob/master/Capstone_part2.ipynb

https://medium.com/@himanshu_23732/sentiment-analysis-with-sentiment140-e6b0c789e0

http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81435/6/jsobrinosTFM0618memoria.pdf

Modelos Avanzados

https://code.google.com/archive/p/word2vec/

https://medium.com/swlh/sentiment-classification-using-word-embeddings-word2vec-ae

https://en.wikipedia.org/wiki/Long_short-term_memory