

# Análisis de sentimiento de tuits en español mediante una red neuronal convolucional

Diego Guevara Acuña B02846<sup>1</sup> y Vladimir Sagot Muñoz B15919<sup>2</sup>

*CI-1441 Paradigmas Computacionales*

*Escuela de Ciencias de la Computación e Informática*

*Universidad de Costa Rica*

<sup>1</sup>*diego.guevara@ucr.ac.cr*, <sup>2</sup>*vladimir.sagot@ucr.ac.cr*

Noviembre, 2019

## Resumen

Se presenta un prototipo de un sistema de análisis de sentimiento de tuits en español, utilizando como medio una red neuronal. Esta fue entrenada con tuits clasificados como positivos o negativos. A través del modelado de una red neuronal convolucional, implementada en Python, se desarrolló la investigación alrededor de la precisión obtenida al usar este método de aprendizaje y clasificación. Se realizaron pruebas con un dataset de tuits y se graficaron los resultados de sus predicciones. Se concluye que una red neuronal convolucional puede analizar el sentimiento de tuits con una buena precisión.

**Palabras clave:** *Red neuronal, red neuronal convolucional, tuit, analisis de sentimientos*.

## 1. Introducción

Este artículo presenta el trabajo desarrollado como proyecto del curso CI-1441 Paradigmas Computacionales. La asignación tuvo como fin ayudar a entender los principios metodológicos y los problemas fundamentales de la Inteligencia Artificial a través de métodos de aprendizaje y clasificación. Para este proyecto se escogieron a las redes neuronales como método para ser desarrollado.

Como justificación, el análisis de sentimiento ha tomado mucha importancia en el monitoreo de redes

sociales para aproximarse con seguridad a la opinión pública sobre ciertos temas. Se utiliza por ejemplo para poder cuantificar la opinión de muchos usuarios de un servicio en poco tiempo.

El proyecto propuesto consistió en analizar los sentimiento de tuits en idioma español utilizando una red neuronal, la cual fue entrenada empleando tuits previamente clasificados como positivos o negativos. Se escogió desarrollar este tema en su versión español, dado que la mayoría de investigaciones similares se encuentran en inglés y extender la misma a nuestro idioma nativo le agregó un componente de innovación al proyecto.

En la sección 2 se describe la pregunta que se quiso responder con este trabajo. En la sección 3 se explica el problema que se trató. La sección 4 presenta el estado del arte con artículos en los cuales se han realizado trabajos similares. La sección 5 trata el marco teórico del proyecto. La sección 6 contiene el objetivo general del trabajo y, además, los objetivos específicos. La sección 7 describe de forma detallada la metodología y los pasos que se siguieron durante el desarrollo del proyecto. La octava sección explica la implementación de la red y los experimentos realizados. En la parte 9 se analizan los resultados obtenidos. La sección 10 incluye las conclusiones a las que se llegaron luego de realizar el proyecto. Por último, se presenta la bibliografía consultada.

## 2. Pregunta de la Investigación

¿Qué tipo de red neuronal artificial puede contribuir a determinar si es posible analizar el sentimiento de tuits en español de una forma precisa?.

## 3. Problema del Proyecto

Construir un prototipo de red neuronal capaz de ser entrenado para clasificar como positivo o negativo el sentimiento expresado en distintos tuits.

## 4. Estado del Arte

El análisis de sentimientos en textos de redes sociales es un tema ampliamente estudiado y con distintos métodos propuestos para ser resuelto a través de la Inteligencia Artificial. Ante la diversidad de escenarios propuestos, todos en inglés, el principal reto es encontrar uno que ofrezca simplicidad, efectividad y que se pueda modelar para el idioma español.

Recientemente se ha comparado el rendimiento de distintos algoritmos para el análisis de emociones a través de una red neuronal convolucional (7). Destacando el alto grado de éxito del uso de redes neuronales de dos capas y, que los mejores resultados, se han obtenido utilizando una capa de tipo Softmax antes de la salida de la red neuronal.

Existen modelos nuevos de análisis de sentimientos en tuits (1) que proponen que el mismo debe ir más allá de los caracteres por analizar. En tanto la precisión del sentimiento sobre un mensaje se debe afinar incluyendo la información del perfil de una persona, es decir, el contexto de quien emite el tuit. Esta investigación agrega un grado de complejidad al modelo del análisis de sentimientos y abre variables importantes por analizar en el entrenamiento de una red neuronal y la privacidad de la información de las personas.

Apegado al objetivo de diseñar una solución que ofrezca simplicidad y efectividad y considerando el alto procesamiento de datos que suponen las redes sociales, parece que los modelos complejos de análisis de sentimiento en redes sociales topan con barreras

para su desarrollo (6). Un objetivo importante, entonces, es lograr simplificar un texto, eliminar ruido dentro del mismo y de esta forma lograr entradas para una red neuronal que genere resultados precisos.

Tomando en cuenta lo señalado por las investigaciones, se tomó en cuenta para este proyecto la consideración de un proceso de eliminación de ruido en los tuits por analizar, así como la estructura de una red neuronal convolucional de dos capas. Ambas características cumplen con el objetivo de buscar un modelo de análisis de sentimientos con simplicidad y efectividad.

## 5. Marco Teórico

### 5.1. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural o NLP (*Natural Language Processing*) es una disciplina que incluye áreas como las Ciencias de la Computación, la Inteligencia Artificial y Psicología Cognitiva. Su objetivo principal es que las máquinas sean capaces de comprender los idiomas con los que los humanos se comunican. Además, esta disciplina trata de explicar cómo las personas comprenden el significado de una oración hablada o escrita y cómo diferencian entre una suposición, una creencia o un hecho (4).

En el Procesamiento de Lenguaje natural se realizan distintos tipos de análisis:

- **Análisis Fonético:** se refiere a la pronunciación de las palabras. Es importante cuando se procesan palabras habladas, no en texto.
- **Análisis Morfológico:** significa estudiar cómo se construye la estructura de las palabras para clasificarlas.
- **Análisis Semántico:** se busca entender el significado de las oraciones. Las palabras pueden tener distintos significados dependiendo del contexto de la oración.
- **Análisis Sintáctico:** Se refiere a la acción de dividir las oraciones en cada uno de sus componentes.

## 5.2. Simplificación de texto

Consiste en transformar un texto en un equivalente que es más fácil de entender para una audiencia determinada (Bautista S.). Esto se realiza dividiendo las oraciones complejas en oraciones más simples y reemplazando el vocabulario complejo por un vocabulario común.

Existen diferentes maneras de medir la facilidad de lectura de un texto basadas en ecuaciones matemáticas que utilizan métricas como número de pronombres personales en el texto, el número de sílabas por palabra o el número de palabras por oración.

## 5.3. Redes neuronales

Representan un modelo computacional de Inteligencia Artificial que trata de simular el funcionamiento del cerebro humano a través de entradas de información, capas de neuronas y salidas de información. Estas logran adquirir cierta “inteligencia” (capacidad de reconocer patrones) a través de etapas de entrenamiento llamadas “épocas” (3).

Cobran relevancia para el presente proyecto ya que el análisis de tuits en redes sociales representa un reto con gran volumen de información. Lo cual permite tener bastantes entradas de información para entrenar y probar este modelo, así como gran variedad de patrones para orientar el posible sentimiento expresado en un texto.

# 6. Objetivos

## 6.1. Objetivo general

Determinar la posibilidad de una red neuronal artificial para clasificar como positivo o negativo el sentimiento expresado en tuits.

## 6.2. Objetivos específicos

- Realizar una revisión bibliográfica sobre el análisis de sentimientos en texto a través de las redes neuronales para determinar una solución óptima al problema.

- Diseñar un modelo de red neuronal artificial que permita clasificar como negativo o positivo el sentimiento en tuits.
- Analizar la precisión de los resultados generados por el modelo de red neuronal artificial diseñado.

# 7. Metodología

Para lograr cumplir con los objetivos propuestos en este proyecto, se propusieron las siguientes tareas:

- Investigar sobre el funcionamiento de las redes neuronales artificiales.
- Estudiar el funcionamiento de las aplicaciones de redes neuronales para el análisis de sentimientos en texto.
- Definir los requerimientos funcionales y no funcionales de prototipo de red neuronal que se desea crear.
- Implementar un prototipo de la red neuronal que se quiere crear. Para implementarlo se ha decidido utilizar el lenguaje de programación Python.
- Entrenar el prototipo de red neuronal con un dataset de tamaño suficientemente grande. Para esta tarea se ha seleccionado un dataset de 177748 tuits en Español que consta de 55531 tuits positivos y 122217 tuits negativos. Este dataset se encuentra disponible en: [github.com/garnachod/TwitterSentimentDataset](https://github.com/garnachod/TwitterSentimentDataset).
- Registrar el funcionamiento y resultados del prototipo, para ser evaluados posteriormente.
- Si se detectan fallos o errores, aplicar las correcciones requeridas.
- Generar conclusiones sobre los resultados obtenidos.

Para facilitar las tareas que involucran implementar, entrenar y probar el prototipo se utilizó un ambiente Linux para implementar la red y se utilizarán diferentes paquetes y herramientas para la construcción de redes neuronales en Python.

La biblioteca de Python que se usa de forma principal es el API Keras, una biblioteca para redes neuronales de código abierto que soporta redes convolucionales y recurrentes. Se requirió además TensorFlow, una biblioteca para aprendizaje desarrollado por Google para sistemas que requieren construir y entrenar redes neuronales. Por último, el modelo GloVe para representación de palabras distribuidas, el cual utiliza un algoritmo para obtener representaciones vectoriales de palabras.

## 8. Implementación de la red

Para programar la red deseada, se investigó sobre trabajos similares y se siguieron tutoriales sobre este tipo de redes. Especialmente, se basó gran parte del proyecto en el trabajo de Usman Malik en su artículo (5). Este artículo presenta como crear una red neuronal similar para críticas de películas.

La red neuronal convolucional implementada<sup>1</sup> cuenta con una capa de entrada correspondiente a un embedding vector, el cual es una matriz de palabras distribuidas por relación semántica y creada con el Spanish Billion Word Corpus<sup>2</sup>. Esta acepta un máximo de 140 palabras como entrada para la RNA.

De segundo la RNA se compone de una capa de convolución que reduce significativamente la entrada original y el número de parámetros a procesar por parte de esta, de 5 millones de parámetros a 230.000. La capa número tres realiza un proceso de pooling, como se muestra en la siguiente figura, reduce a una dimensión la entrada de datos de la RNA.

Por último, antes de la capa de salida, se agregó una capa de 128 neuronas que funcionan con una función de activación de tipo relu. La cual es clave para el resultado final, ya que sin esta el porcentaje de éxito se reducía de forma significativa.

<sup>1</sup>Se adjunta el código utilizado con el artículo.

<sup>2</sup>Spanish Billion Word Corpus and Embeddings: <https://crscardellino.github.io/SBWCE/>

```
Creating train and test data...
Classification with Convolutional Neural Network...
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 140, 300)	5214900
conv1d_1 (Conv1D)	(None, 135, 128)	230528
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 1)	129

```
Total params: 5,462,069
Trainable params: 247,169
Non-trainable params: 5,214,900
>>>
```

Figura 1: Red neuronal implementada.

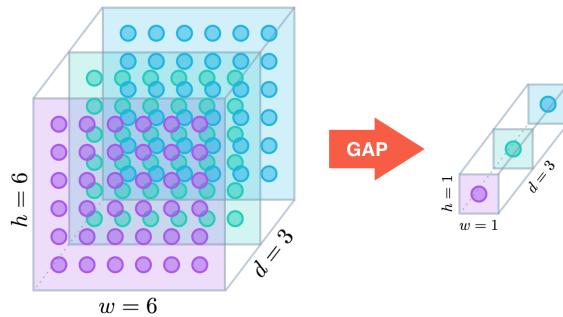


Figura 2: Proceso de pooling de la RNA.

## 9. Análisis de resultados

Se procedió a entrenar la RNA con un 80 % de set de datos escogido de forma aleatoria y con cinco iteraciones, por lo que el 20 % restante se utilizó para ser probado. Después de distintos experimentos y ajustes para afinar los resultados, se logró obtener poco más de 80 % de éxito en el análisis de sentimiento de los tuits analizados.

Como se muestra en la figura 3, en las distintas cinco épocas escogidas se logró optimizar la precisión de la RNA. Además, se logró mantener un resultado constante en la precisión durante la etapa de prueba de la red.

Es importante mencionar que la ejecución de las pruebas se hizo con una muestra de 11.000 tuits del set de datos, para lograr una ejecución en tiempo prudencial de la RNA. Que a la misma vez permitiera ajustar cambios para mejorar la precisión de la misma.

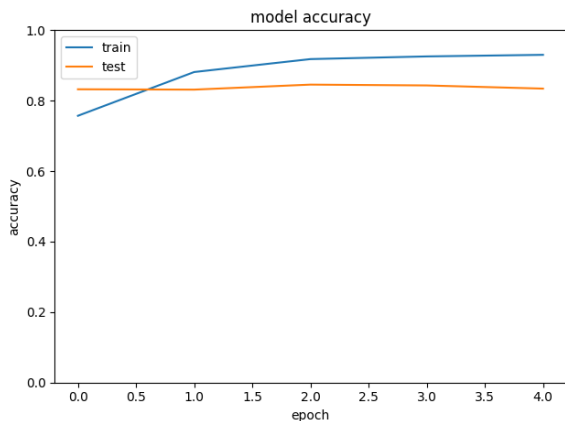


Figura 3: Resultado de la etapa de entrenamiento y prueba de la RNA.

## 10. Conclusiones

Con respeto a los objetivos iniciales de proyecto, se logró cumplir con todos de manera satisfactoria. Ya que según los resultados brindados, determinamos que sí es posible que una RNA determine el sentimiento en un tuit de forma casi precisa, con un margen de acierto del 80 %.

De la misma forma se logró abarcar distintas fuentes bibliográficas que ayudaron a desarrollar un modelo de RNA, el cual se implementó con código en Python y brindó los resultados que se esperaban bajo el objetivo general.

Una red neuronal convolucional puede clasificar los sentimientos en tuits en español de forma ágil, cuando se posea un set de datos amplio y un contexto determinado donde se utilice la misma clase de español.

## A. Anexos

### A.1. Manual de uso

El siguiente es un manual de uso que explica de forma breve cómo ejecutar el experimento realizado en este artículo.

1. El archivo ***tweets-step1.py*** es el encargado de generar la matriz de embedding words a partir de un diccionario más amplio. Es importante crear la misma, para que exista una matriz con únicamente las palabras utilizadas en el set de datos. Esto permite generar una entrada a la RNA con el tamaño necesario al set de datos utilizado. Este código guarda en un archivo el objeto creado, el cual posteriormente se procede a cargar, como explica el siguiente paso.
2. Una vez creada la matriz de embedding words el archivo ***tweets-convolutional-neural-network.py*** es el que corre la RNA y realiza los experimentos necesarios. Este carga el objeto creado en un objeto con el código anterior. Este es un paso creado de forma opcional para ahorrar tiempo de ejecución durante cada ejecución de código.
3. Para la ejecución de este código solo basta tener los archivos a cargar en el mismo directorio que el código que se ejecuta.
4. Se recomienda ser ejecutado con el IDE de Net-brains, PyCharm. Ya que ayuda a instalar de forma automatizada las bibliotecas requeridas y muestra de forma correcta el plot generado al final de la ejecución del código de la RNA.

### A.2. Glosario sobre redes neuronales

Los siguientes conceptos dan un marco de referencia sobre el funcionamiento de las redes neuronales y sus elementos (3):

- Red neuronal artificial: Comúnmente se le denomina solo “red neuronal”. Es un modelo computacional de Inteligencia Artificial que trata de simular el funcionamiento del cerebro humano a

través de entradas de información, capas de neuronas y salidas de información. Logra adquirir cierta inteligencia a través de etapas de entrenamiento con información dada.

- Red neuronal artificial: Comúnmente se le denomina solo “red neuronal”. Es un modelo computacional de Inteligencia Artificial que trata de simular el funcionamiento del cerebro humano a través de entradas de información, capas de neuronas y salidas de información. Logra adquirir cierta inteligencia a través de etapas de entrenamiento con información dada.
- Neurona: Es la unidad fundamental de funcionamiento de una red neuronal. Cuentan con una o más entradas de información y una función de activación que usualmente busca un resultado normalizado para generar una salida.

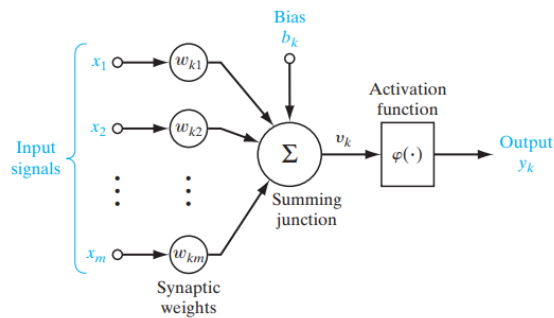


Figura 4: Ejemplo gráfico de una red neuronal.

- Función de activación: Es la que determina la salida de una neurona. Necesita tomar la entrada a través de un proceso de sumatoria de las entradas de una neurona y luego aplicar una función de tipo umbra o sigmoidea. Su resultado determina la información que se pasa a neuronas en otra capa o como resultado final.
- Capas: Determinan el nivel de profundidad de una red neuronal, ya que una red puede contener una única capa de neuronas dentro o varias.
- Red neuronal como grafo: Una red neuronal se representa como un grafo, donde cada capa tiene

neuronas que se conectan con la capa anterior y la capa siguiente. Los enlaces son el equivalente a los pesos en un grafo y pueden tener pesos que determinan el resultado de la función de activación, según la importancia de una de las entradas de una neurona.

- Enlaces sinápticos: Funcionan como un peso con entrada y salida de tipo lineal, es decir, el peso multiplica a la entrada siempre.
- Enlaces de activación: Funcionan como un peso con entrada y salida de tipo no lineal, es decir, el peso tiene como valor una función que determina el resultado.
- Propagación: Es el proceso mediante el cual una red neuronal ajusta sus valores y se entrena. Se utiliza cálculo matemático para ajustar los valores de los pesos de la red neuronal, para que la información con la que se entrena la red se ajuste con las entradas y salidas dadas.
- Red neuronal convolucional: Es un tipo de red neuronal de tipo multicapa que busca simplificar la información que analiza en cada capa, a través de técnicas estadísticas. De esta forma logra resultados en una forma más precisa y óptima.

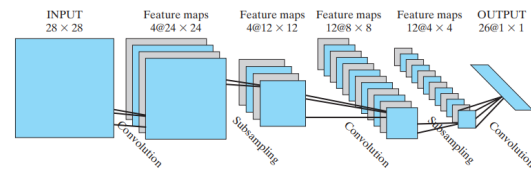


Figura 5: Ejemplo de una red neuronal convolucional.

## Referencias

- [1] Alharbi, A. S. M. and de Doncker, E. (2019). Twitter sentiment analysis with a deep neural network: An enhanced approach using user behavioral information. *Cognitive Systems Research*, 54:50 – 61.

- [Bautista S.] Bautista S., G. P. *Simplificación de texto para facilitar la comprensión lectora del usuario final*.
- [3] Haykin, S. S. (2009). *Neural networks and learning machines*. Pearson Education (US).
- [4] López, R. (2017). Procesamiento del lenguaje natural con python.
- [5] Malik, U. (2019). Python for nlp: Movie sentiment analysis using deep learning in keras.
- [6] Severyn, A. and Moschitti, A. (2015). UNITN: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 464–469, Denver, Colorado. Association for Computational Linguistics.
- [7] Stojanovski, D., Strezoski, G., Madjarov, G., Dimitrovski, I., and Chorbev, I. (2018). Deep neural network architecture for sentiment analysis and emotion identification of twitter messages. *Multi-media Tools Appl.*, 77(24):32213–32242.

### A.3. Código en Python de la red neuronal convolucional implementada

```
1 import matplotlib.pyplot as plt
2 import nltk
3 import numpy as np
4 import pandas as pd
5 import pickle
6 import random
7 import re
8 import seaborn as sns
9 import tensorflow as tf
10
11 import os
12
13 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
14
15 from nltk.corpus import stopwords
16 from numpy import array
17 from keras.preprocessing.text import one_hot
18 from keras.preprocessing.sequence import pad_sequences
19 from keras.models import Sequential
20 from keras.layers.core import Activation, Dropout, Dense
21 from keras.layers import Flatten, Conv1D, LSTM
22 from keras.layers import GlobalMaxPooling1D
23 from keras.layers.embeddings import Embedding
24 from sklearn.model_selection import train_test_split
25 from keras.preprocessing.text import Tokenizer
26 from numpy import array
27 from numpy import asarray
28 from numpy import zeros
29
30
31 def preprocess_text(sen):
32     # Transform to lowercase
33     sentence = sen.lower()
34
35     # Removing URLs
36     sentence = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\(\)]|(?:%[0-9a-fA-F]
37     [0-9a-fA-F]))+', ' ', sentence)
38
39     # Removing username mentions "@"
40     sentence = re.sub(r'@[a-zA-Z0-9_]*', ' ', sentence)
41
42     # Remove punctuations and numbers
43     sentence = re.sub('[^a-zA-Z ]', '', sentence)
44
45     # Single character removal
46     sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)
47
48     # Removing multiple spaces
49     sentence = re.sub(r'\s+', ' ', sentence)
50
51     return sentence
52
53 # Define variables
54 # vocab_size = len(tokenizer.word_index) + 1
```



```

55 # vocab_size = 17383
56 # Tweet old size of characters
57 maxlen = 140
58
59 # -----
60 # Load and sanitize data
61 # -----
62
63 print("Load tweets...")
64
65 # Load the tweets dataset
66 tweet_reviews = pd.read_csv("/home/vladimir/Desktop/dataset.csv")
67
68 # Clean dataset of tweets
69 X = []
70 sentences = list(tweet_reviews['tweet'])
71 for sen in sentences:
72     X.append(preprocess_text(sen))
73
74 # Transform positive sentiment to '1' and negative to '0'
75 y = tweet_reviews['class']
76 y = np.array(list(map(lambda x: 1 if x == "pos" else 0, y)))
77
78 print("Creating train and test data...")
79
80 # Create train and test data in a random order
81 # For this case 80% of data is for training and 20% for testing
82 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
83
84 # -----
85 # Embedding layer
86 # -----
87 # Transform textual data into numeric data for the first layer in the deep learning models
    in Keras
88
89 # Vectorize a text corpus, by turning each text into either a sequence of integer
90
91 # Create a word-to-index dictionary
92 # The most common words will be kept
93 tokenizer = Tokenizer(num_words=21000)
94
95 # Updates internal vocabulary based on a list of texts. This method creates the vocabulary
    index based on word
96 # frequency. So if you give it something like, "The cat sat on the mat." It will create a
    dictionary s.t. word_index[
97 # "the"] = 1; word_index["cat"] = 2 it is word -> index dictionary so every word gets a
    unique integer value.
98 # https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method-exactly-do
99 tokenizer.fit_on_texts(X_train)
100
101 # Transforms each text in texts to a sequence of integers. So it basically takes each word
    in the text and replaces
102 # it with its corresponding integer value from the word_index dictionary.
103 X_train = tokenizer.texts_to_sequences(X_train)
104 X_test = tokenizer.texts_to_sequences(X_test)
105
106 # Adding 1 because of reserved 0 index

```

```

107 vocab_size = len(tokenizer.word_index) + 1
108
109 # Tweet old size of characters
110 maxlen = 140
111
112 # pad_sequences is used to ensure that all sequences in a list have the same length
113 # padding: String, 'pre' or 'post': pad either before or after each sequence
114 # https://stackoverflow.com/questions/42943291/what-does-keras-io-preprocessing-sequence-
    pad-sequences-do
115 X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
116 X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
117
118 # -----
119 # Open embedding matrix from file
120 # -----
121 file = open('embedding_matrix.obj', 'rb')
122 embedding_matrix = pickle.load(file)
123 file.close()
124
125 # -----
126 # Text Classification with Simple Neural Network
127 # -----
128
129 print("Classification with Convolutional Neural Network...")
130
131 model = Sequential()
132
133 embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=
    maxlen, trainable=False)
134 model.add(embedding_layer)
135 model.add(Conv1D(128, 6, activation='exponential'))
136 # model.add(Conv1D(64, 6, activation='exponential'))
137 model.add(GlobalMaxPooling1D())
138 model.add(Dense(128, activation='relu'))
139 model.add(Dense(1, activation='sigmoid'))
140 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
141
142 print(model.summary())
143
144 print("Training neural network...")
145
146 # Model train
147 # fit: Trains the model for a fixed number of epochs (iterations on a dataset)
148 # batch_size: Integer or None. Number of samples per gradient update
149 # verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line
    per epoch
150 # validation_split: Float between 0 and 1. Fraction of the training data to be used as
    validation data
151 #
152 # The batch size defines the number of samples that will be propagated through the network.
153 #
154 # For instance, let's say you have 1050 training samples and you want to set up a
    batch_size equal to 100. The
155 # algorithm takes the first 100 samples (from 1st to 100th) from the training dataset and
    trains the network. Next,
156 # it takes the second 100 samples (from 101st to 200th) and trains the network again. We
    can keep doing this

```

```

157 # procedure until we have propagated all samples through of the network.
158 # https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network
159
160 history = model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split
161                     =0.2)
162
163 score = model.evaluate(X_test, y_test, verbose=1)
164
165 print("Test Score:", score[0])
166 print("Test Accuracy:", score[1])
167
168 plt.plot(history.history['acc'])
169 plt.plot(history.history['val_acc'])
170
171 plt.title('model accuracy')
172 plt.ylabel('accuracy')
173 plt.xlabel('epoch')
174 plt.legend(['train', 'test'], loc='upper left')
175 plt.ylim(0, 1)
176 plt.show()
177
178 plt.plot(history.history['loss'])
179 plt.plot(history.history['val_loss'])
180
181 plt.title('model loss')
182 plt.ylabel('loss')
183 plt.xlabel('epoch')
184 plt.legend(['train', 'test'], loc='upper left')
185 plt.ylim(0, 1)
186 plt.show()

```

```

1 import matplotlib.pyplot as plt
2 import nltk
3 import numpy as np
4 import pandas as pd
5 import pickle
6 import random
7 import re
8 import seaborn as sns
9 import tensorflow as tf
10
11 from nltk.corpus import stopwords
12 from numpy import array
13 from keras.preprocessing.text import one_hot
14 from keras.preprocessing.sequence import pad_sequences
15 from keras.models import Sequential
16 from keras.layers.core import Activation, Dropout, Dense
17 from keras.layers import Flatten
18 from keras.layers import GlobalMaxPooling1D
19 from keras.layers.embeddings import Embedding
20 from sklearn.model_selection import train_test_split
21 from keras.preprocessing.text import Tokenizer
22 from numpy import array
23 from numpy import asarray
24 from numpy import zeros
25
26
27 def preprocess_text(sen):

```

```

28     # Transform to lowercase
29     sentence = sen.lower()
30
31     # Removing URLs
32     sentence = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&]|[*\(\)]|(?:%[0-9a-f-F
33     ][0-9a-fA-F]))+', ' ', sentence)
34
35     # Removing username mentions "@"
36     sentence = re.sub(r'@[a-zA-Z0-9_]*', ' ', sentence)
37
38     # Remove punctuations and numbers
39     sentence = re.sub('[^a-zA-Z ]', ' ', sentence)
40
41     # Single character removal
42     sentence = re.sub(r'\s+[a-zA-Z]\s+', ' ', sentence)
43
44     # Removing multiple spaces
45     sentence = re.sub(r'\s+', ' ', sentence)
46
47     return sentence
48
49 # -----
50 # Load and sanitize data
51 # -----
52
53 print("Load tweets...")
54
55 # Load the tweets dataset
56 tweet_reviews = pd.read_csv("/home/vladimir/Desktop/dataset.csv")
57
58 # Clean dataset of tweets
59 X = []
60 sentences = list(tweet_reviews['tweet'])
61 for sen in sentences:
62     X.append(preprocess_text(sen))
63
64 # Transform positive sentiment to '1' and negative to '0'
65 y = tweet_reviews['class']
66 y = np.array(list(map(lambda x: 1 if x == "pos" else 0, y)))
67
68 print("Creating train and test data...")
69
70 # Create train and test data in a random order
71 # For this case 80% of data is for training and 20% for testing
72 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
73
74 # -----
75 # Embedding layer
76 # -----
77 # Transform textual data into numeric data for the first layer in the deep learning models
78 # in Keras
79
80 # Vectorize a text corpus, by turning each text into either a sequence of integer
81
82 # Create a word-to-index dictionary
83 # The most common words will be kept

```

```

83 tokenizer = Tokenizer(num_words=21000)
84
85 # Updates internal vocabulary based on a list of texts. This method creates the vocabulary
86 # index based on word
87 # frequency. So if you give it something like, "The cat sat on the mat." It will create a
88 # dictionary s.t. word_index[
89 # "the"] = 1; word_index["cat"] = 2 it is word -> index dictionary so every word gets a
90 # unique integer value.
91 # https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method-exactly-do
92 tokenizer.fit_on_texts(X_train)
93
94 # Transforms each text in texts to a sequence of integers. So it basically takes each word
95 # in the text and replaces
96 # it with its corresponding integer value from the word_index dictionary.
97 X_train = tokenizer.texts_to_sequences(X_train)
98 X_test = tokenizer.texts_to_sequences(X_test)
99
100 # Adding 1 because of reserved 0 index
101 vocab_size = len(tokenizer.word_index) + 1
102
103 # Tweet old size of characters
104 maxlen = 140
105
106 # pad_sequences is used to ensure that all sequences in a list have the same length
107 # padding: String, 'pre' or 'post': pad either before or after each sequence
108 # https://stackoverflow.com/questions/42943291/what-does-keras-io-preprocessing-sequence-
109 # pad-sequences-do
110 X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
111 X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
112
113 print("Load GloVe spanish file...")
114
115 # In word embeddings, every word is represented as an n-dimensional dense vector. The words
116 # that are similar will
117 # have similar vector. https://stackabuse.com/python-for-nlp-word-embeddings-for-deep-
118 # learning-in-keras/
119 #
120 # Data source in Spanish: https://github.com/dccuchile/spanish-word-embeddings#glove-
121 # embeddings-from-sbwc
122 # Algorithm: Implementation: GloVe
123 # Parameters: vector-size = 300, iter = 25, min-count = 5, all other parameters set as
124 # default
125 embeddings_dictionary = dict()
126 glove_file = open('/home/vladimir/Downloads/Compressed/glove-sbwc.i25.vec', encoding="utf8"
127 )
128
129 for line in glove_file:
130     # splits a string into a list
131     records = line.split()
132     # the word is in position 0
133     word = records[0]
134     # get n-dimensions of array
135     vector_dimensions = asarray(records[1:], dtype='float32')
136     embeddings_dictionary[word] = vector_dimensions
137 glove_file.close()
138
139 # Save embeddings_dictionary into a file

```

```

130 file = open('embeddings_dictionary.obj', 'wb')
131 pickle.dump(embeddings_dictionary, file)
132 file.close()
133
134 print("Creating embedding matrix...")
135
136 # Creates an embedding matrix where each row number will correspond to the index of the
    word in the corpus
137 # The matrix will have 300 columns where each column will contain the GloVe word embeddings
    for the words in our corpus
138 # Create a matrix with zeros
139 embedding_matrix = zeros((vocab_size, 300))
140 for word, index in tokenizer.word_index.items():
141     embedding_vector = embeddings_dictionary.get(word)
142     if embedding_vector is not None:
143         embedding_matrix[index] = embedding_vector
144
145 # Save embedding_matrix into a file
146 file = open('embedding_matrix.obj', 'wb')
147 pickle.dump(embedding_matrix, file)
148 file.close()

```