***Trabajo Práctico***

***N°2***

***“Predicciones de un set de datos”***

* **Carrera:** Ingeniería en Informática.
* **Materia:** Organización de Datos.
* **Profesor:** Argerich, Luis
* **JTP**: Golmar, Natalia.
* **Cuatrimestre**: 1er Cuatrimestre 2020.
* **Fecha de entrega:** 21/05/2020
* **Nombre del grupo:** *Team\_Undav*
* **Integrantes:**
  + - * Calonge, Federico Matías.
      * Ceballos Pardo, Sarah.
      * Flores, Matías.
      * Loiseau, Matías.

# Índice

1-Objetivo. (Pág. )

2- Introducción: Conceptos previos. (Pág. )

2.1-NLP. (Pág. )

2.1.1-Definición. (Pág. )

2.1.2-Aplicaciones. (Pág. )

2.1.3-Librerias de NPL en Python. (Pág. )

2.2-Conceptos NLP. (Pág. )

2.2.1-Corpus. (Pág. )

2.2.2-Stop y Rare Words. (Pág. )

2.2.3-Tokenización. (Pág. )

2.2.4-POS (Parts of Speech) Tagging (Pág.)

2.2.5-Stemming o lematización (Pág. )

2.3-Modelos para extraer features/vocabulario de nuestros CORPUS. (Pág. )

2.3.1-Bag of Words (BOW) (Pág. )

2.3.2-TF-IDF. (Pág. )

2.3.3-N-GRAMS y Feature Hashing. (Pág. )

2.3.4- Word Embeddings. (Pág. )

2.3.4.1-Funcionamiento Word2Vec para obtener

nuestros Word Embeddings. (Pág. )

2.4-Cómo Clasificar Texto. (Pág.)

Acá ponemos solo los que usamos → **2.4.1-Naive Bayes.** (Pág. )

**2.4.2-CNN.** (Pág. )

**2.4.3-LSTM.** (Pág. )

**2.4.4-Transformers.** (Pág. )

**2.4.5-BERT.** (Pág. )

3- Desarrollo. (Pág. 15)

3.1-Data set. (Pág. )

3.2- Resultados y análisis de Tweets - TP1 (Pág. )

3.3-Preprocesamiento de Texto. (Pág. )

**3.4-Obtención de nuestro Modelo y Predicciones de Tweets.** (Pág. )

**3.2.1- VER METODO MAS BASICO → BoW y NAIVE BAYES?**

**3.2.2-VER USANDO EMBEDDINGS ENTRENADOS CON**

**NUESTROS TEXTOS y predicciones con… VER.**

**3.2.3-USANDO EMBEDDINGS YA ENTRENADOS y predicciones con… VER.**

**3.2.4- OTRO METODO → CNN Y LSTM SEPARADOS?**

**3.2.5- OTRO MÉTODO → CNN Y LSTM JUNTOS?**

**3.2.6- OTRO METODO ¿TRANSFORMER?**

**3.2.7- BERT.**

3.3-Resultados finales.

4-Conclusiones. (Pág. 30)

5-Mejoras. (Pág. 32)

6-Bibliografía. (Pág. 33)

**1-Objetivo**

El objetivo de este Trabajo Práctico N°2 consiste en …………….

Dicho dataset lo obtenemos de la siguiente competencia: <https://www.kaggle.com/c/nlp-getting-started>

**Para realizar dichas predicciones anteriormente tuvimos que realizar un análisis exploratorio del data set, el cual fue realizado en el TP N°1; donde** buscamos comprender y entender los datos mediante distintos análisis estadísticos que aplicaremos sobre los datos de nuestro dataset. De esta manera pudimos encontrar relaciones entre nuestras columnas / atributos y obtener resultados en base a visualizaciones que nos permitieron transmitir información que no es fácilmente apreciada por las personas cuando se observan los datos de una manera más “cruda”.

En este Trabajo Práctico N°2, se buscará…..

# 

# 2-Introducción y conceptos previos

En esta Sección describiremos los temas teóricos y matemáticos para tratarlos a lo largo del desarrollo del Informe y llevarlos a cabo mediante los Algoritmos de Python en el ambiente de Anaconda.

## 2.1-NLP

**2.1.1-Definición**.

NLP (Natural Language Processing o Procesamiento del Lenguaje Natural) es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. Su objetivo principal es desarrollar algoritmos que permitan a los sistemas/ aplicaciones/ máquinas comprender textos y comprender el lenguaje natural/humano.

Para abordar este problema, se comenzaron a desarrollar diversas aplicaciones utilizando enfoques de aprendizaje automático. Este objetivo se puede lograr recolectando una gran cantidad de texto y luego entrenando al algoritmo para realizar diversas tareas como categorizar texto y/o modelar temas.

Gracias a los millones de GB y TB diarios generados por blogs, sitios web sociales y páginas web hoy en día es muy beneficioso e interesante usar NLP. Hay muchas empresas que reúnen todos estos datos para comprender a los usuarios y sus pasiones y dar estos informes a las empresas para ajustar sus planes.

Estos datos podrían mostrar que la gente de Brasil está contenta con el producto A, que podría ser una película o cualquier cosa, mientras que la gente de EE. UU. está contenta con el producto B. Y esto podría ser instantáneo (resultado en tiempo real). Al igual que los motores de búsqueda, dan los resultados adecuados a las personas adecuadas en el momento adecuado.

Podemos resolver muchísimos problemas con NPL, tales como:

* Clasificación de textos (puede ser en muchas categorías o binaria). Un ejemplo de clasificación binaria es hacer análisis de sentimiento (decir por ej. si un review es positivo -1- o negativo -0-).
* Entity Recognition: esto es reconocer entidades en los textos y tagearlas (por ejemplo personas, empresas, fechas, lugares, etc.).
* Topic modeling: de esta manera podemos entender cuáles son los temas principales de un texto y extraerlos.
* Y muchísimos más.

**2.1.2-Aplicaciones.**

Podemos encontrar muchísimas aplicaciones dentro del mundo de NLP:

* Motores de búsqueda: como Google.
* Feeds de los sitios web sociales: como la fuente de noticias de Facebook. El algoritmo de feed de noticias entiende tus intereses mediante NLP y muestra anuncios y publicaciones relacionadas con mayor probabilidad que otras publicaciones.
* Interfaces de conversación (reconocimiento de voz y traducción del habla): por ejemplo el motor de voz de Apple, Siri.
* Procesadores de documentos: NLP permite comprender oraciones completas y hasta incluso escribir oraciones y párrafos completos gramaticalmente correctos.
* Filtros de Spam: por ejemplo los filtros de spam de Gmail: No se trata solo del filtro de spam habitual, ahora los filtros de spam entienden qué hay dentro del contenido del correo electrónico y ver si es un correo no deseado o no.
* Traducción de textos automática.
* Resúmenes automáticos: dado un texto nos devuelve el resumen condensado.
* Modelos de recomendación.
* Y muchísimas más.

**2.1.3-Librerías de NPL en Python.**

Las librerías de NPL más populares para Python son:

* Natural language toolkit (NLTK): es la biblioteca más de Python más popular y fácil de usar para el procesamiento del lenguaje natural (NLP) https://www.nltk.org/
* Spacy: librería NLP para Python más rápida que NLTK.
* Gensim: librería de Python que nos provee acceso al modelo Word2Vect y otros algoritmos para entrenar y obtener nuestros word embeddings. También nos provee de word embeddings ya pre-entrenados para descargarlos y utilizarlos (los cuales están entrenados con las noticias de Google).
* Apache OpenNLP.
* Stanford NLP suite.
* Gate NLP library.

Estas librerías nos proveen distintas funciones para poder interactuar con el lenguaje humano: tokenizar o lematizar palabras, buscar relaciones entre palabras más similares, buscar sinónimos, antónimos, etc. Para algunas de estas funciones se utilizan modelos de redes neuronales implementados en dichas librerías. En nuestro caso, para realizar análisis y predicciones utilizamos las librerias….. COMPLETAR EN BASE A LAS QUE USAMOS.

## 2.2-Conceptos NLP.

A continuación explicaremos varios conceptos de NLP que tomaremos en cuenta para en análisis de nuestros textos en Python.

**2.2.1-Corpus.**

## También llamado “Corpus Lingüístico”, es un conjunto amplio y estructurado de ejemplos reales de uso de la lengua. En nuestro caso utilizamos como Corpus a …………. COMPLETAR → AUQNUE LOS CORPUS CREO QUE LO USAN LOS ALGORITMOS POR DENTRO, NO SON LOS TWEETS, VER. Estos corpus deben ser un conjunto de textos que deben ser relativamente grande, creado independientemente de sus posibles formas o uso... osea que la estructura, variedad y complejidad del corpus debe reflejar dicha lengua de la forma más exacta posible. La idea es que representen al lenguaje de la mejor forma posible para que los modelos de NLP puedan aprender los patrones necesarios para entender el lenguaje.

## 2.2.2-Stop y Rare Words.

## Stop words: palabras muy frecuentes que tenemos que decidir qué hacer con estas (hay listas de stop words precompiladas que nos dice cuáles son las mismas para un determinado idioma). Por ejemplo: “the”, “a”, “this” en inglés.

## Rare words: palabras “raras” que no aparecen en cinco o más documentos por ejemplo. Estas podríamos querer filtrarlas también como las stop words.

## 2.2.3-Tokenización.

## Cuando tratamos con texto, necesitamos dividirlo en partes más pequeñas para su análisis. Por esto, al tokenizar, lo que estamos haciendo es separando el texto de entrada en entidades más pequeñas o palabras, llamadas tokens, con las que trabajaremos luego. En un tokenizador debemos evaluar muchas cosas: qué hacemos con los signos de puntuación, guiones, underscores, fechas, si utilizamos los signos de puntuación como token o no, si le damos o no importancia a las mayúsculas y si unificamos o no palabras similares en un mismo token.

## Por esto, usar un buen tokenizador es muy importante. No es recomendable hacer nuestros propios tokenizadores, ya hay muchos tokenizadores que funcionan bastante bien y sirven para cualquier idioma.

## Para los análisis y predicciones utilizamos el tokenizador provisto por la librería NLTK (“word\_tokenize”). →CREO QUE NO, EN NUESTRO CAASO EL PROPIO ALGORITMO AL QUE LES PASAMOS LOS TWEETS HACE AUTOMÁTICAMENTE ESTA TOKENIZACIÓN.

**2.2.4-POS (Parts of Speech) Tagging**

POS Tagging es un método de NLP que consiste en ETIQUETAR/TAGEAR a las palabras en sustantivos (noun), adjetivos (adjetives), verbos (verbs), etc. Lo utilizamos …. VER SI UTILIZAMOS… por ej. lo podemos usar para identificar algunos tags y así filtrar las palabras que NO tengan dichos tags; ya que no son informativas para nuestro análisis → VER

## 2.2.5-Stemming o lematización

## Word Stemming es un método de NLP que nos permite convertir palabras en una raíz común (o también llamado “root”/”base”/”stem”). Por ejemplo las palabras "models" y "modeling" provienen/tienen la misma raiz ("model"). De esta manera, podemos considerar dichas palabras (models y modeling) como una sola... como la raiz (model). Aplicaremos esta técnica en…. VER SI LA APLICAMOS.

## 2.3-Modelos para extraer features/vocabulario de nuestros CORPUS

## Podemos usar muchos modelos que utilizan distintos algoritmos/métodos para extraer features de nuestros textos; y así obtener un vocabulario que nos sirva para resolver distintos problemas. De esta manera, los modelos permiten tomar un input de texto y obtener un output numérico de salida, por ejemplo, nos permiten identificar palabras más similares entre sí o poder predecir qué palabra colocar luego de otra en base al contexto.

## Para todos los modelos el resultado final es un VECTOR que representa nuestro texto, el cual lo podremos usar luego en otro modelo de ML para realizar predicciones.

## En nuestro caso…. ver si utilizamos estos modelos para así obtener un Word embedding mediante el entrenamiento del modelo Word2Vec con nuestros tweets o ver si no hacemos nada de esto…. O VER SI HACERLO COMO UN 1ER ACERCAMIENTO!?!?

## Para todos los modelos que nombraremos, debemos tener en cuenta el texto que ingresaremos al mismo y tomar distintas decisiones de cómo indexar y qué hacer con nuestra palabras obtenidas del texto: cómo tokenizar el texto, si tenemos o no en cuenta las Stop y Rare Words, si aplicamos o no POS Tagging y Stemming, etc.

## 

## 

## 

## 

**2.3.1-Bag of Words (BOW)**

Vamos a ver nuestra primera aproximación a cómo extraer features de un Corpus de texto mediante el modelo BoW. BoW es uno de los modelos más simples, nos permite extraer un vocabulario/léxico de nuestro texto. Vocabulario = todos los términos que intervienen en el texto.

Luego de tener este vocabulario lo que vamos a hacer es crear por cada texto un vector binario de un tamaño muy grande, donde cada componente del vector va a ser un 0 o un 1 según aparezca la palabra en el texto. De esta manera nuestro texto estará representado por un vector binario.

El principal problema de BoW es que si tenemos miles de términos entonces nuestro vector tendrá miles de columnas. Pero la ventaja es su sencillez. Este modelo de BoW es usable siempre y cuando tengamos una cantidad de columnas manejable.

## 2.3.2-TF-IDF

Utilizar TF-IDF es el primer “refinamiento” / variante que se nos puede ocurrir aplicar sobre BoW.

* TF = term frequency. Fórmula para TF ftd (frecuencia del término en el documento cuántas veces aparece). Hay otras fórmulas que implican aplicar log a ftd o aplicar log a ftd.
* IDF = inverse document frequency. Fórmula para IDF log (N / ft ) logaritmo de la cantidad de documentos “N” / la cantidad de documentos en donde aparece el término (“ft”).

Entonces ahora, a nuestro vector de BoW lo que hacemos es que envés de que esté formado por 1s y 0s... que esté formado por:

* Si el término está en el documento el valor será de TF x IDF.
* Y si no está va a ser 0.

**2.3.3 -N-GRAMS y Feature Hashing.**

El modelo de N-gramas consiste en contabilizar las frecuencias de combinaciones de N palabras. Con N=1 se trata simplemente de contar las frecuencias de las palabras tal y como hicimos antes. Con N=2 contamos las frecuencias de las palabras de acuerdo a la palabra anterior, es decir usando una palabra como contexto. Con N=3 usamos dos palabras

como contexto y así sucesivamente.

Al modelar N-GRAMAS, ahora en nuestro vector no solo vamos a tener UNI-GRAMAS (1-GRAMS) también vamos a tener BI-GRAMAS (2-GRAMS), TRI-GRAMAS (3-GRAMS) y demás. Por ejemplo, para el texto “**The quick brown fox jumped over the lazy dog**” los bi-gramas son los siguientes:

* **The quick.**
* **Quick Brown.**
* **Brown fox.**
* **Fox jumped**
* **Jumped over**
* **Over the**
* **The lazy**
* **Lazy dog.**

Este vector de N-gramas puede ser MUY largo, entonces podemos elegir cuales son los N-grams que vamos a utilizar ahí. De esta manera podemos excluir los más raros que aparecen en muy pocos textos y excluyendo los más populares porque NO nos interesan.

Podriamos aplicar una mejora a esto mediante Feature hashing. De esta manera, lo que hacemos es fijar una cantidad de K columnas fija. Entonces, tanto los unigramas como los bi-gramas, tri-gramas o lo que queramos utilizar... los vamos a HASHEAR en un número entre 0 y K-1. Con este esquema podemos solucionar el problema que teníamos de tener un vector muy largo. Y de esta manera, en un vector de tamaño fijo podemos representar n-gramas en una forma más compacta.

## 

## 

**2.3.4- Modelo Word2Vec y Word Embeddings.**

Los Word Embeddings son una de las variantes más populares para representar textos y extraer features a través de nuestros textos. Un embedding es un VECTOR que representa a una palabra. Si nuestro texto tiene N palabras, entonces tendremos N vectores embeddings para representar a nuestro texto. Estos vectores son DENSOS (NO son dispersos como teníamos en los modelos basados en BoW).

Estos word embeddings son una de las representaciones de un documento de vocabulario más popular. Son vectores "one-hot vector" y son previamente entrenados; de esta manera son capaces de capturar los contextos de una palabra en el documento, similitud semántica y sintáctica, relaciones con otras palabras, etc.

Para obtener/construir estos vectores embeddings hay muchas alternativas... las 3 más populares hoy en día son estos modelos:

**o Word2Vec.**

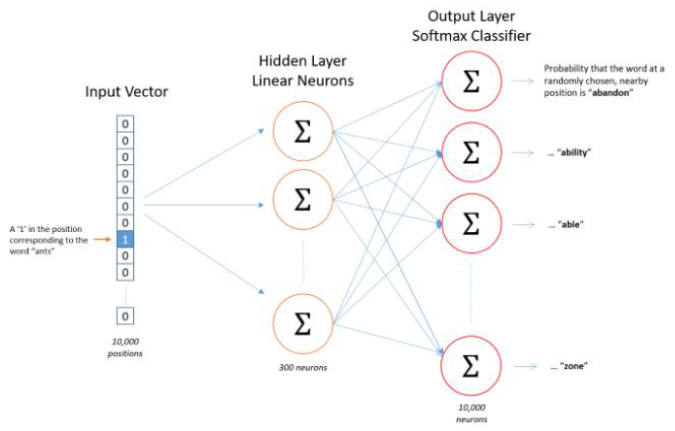
o Glove.

o Starspace.

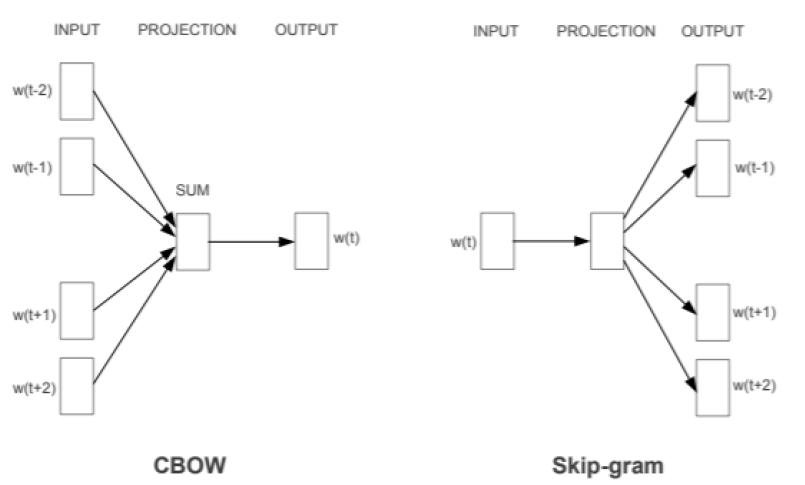
En estas 3 alternativas estos modelos de embeddings se ENTRENAN mediante ANNs. Hay 2 formas para entrenar y obtener estos modelos:

1. CBOW (Continuous Bag Of Words): Se trata de predecir la palabra target de un texto en base a las palabras más cercanas/vecinas (en base al contexto).
2. SKIPGRAMS : **Opuesto a CBOW**. En base a la palabra se trata de predecir cuales son los vecinos (contexto)

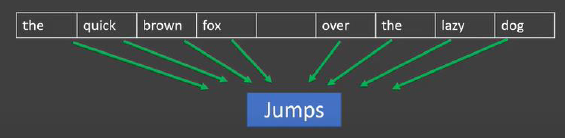
De esta manera con CBOW o SKIPGRAMS entrenamos los distintos **modelos**; y en definitiva vamos a obtener un **embedding** por cada palabra.



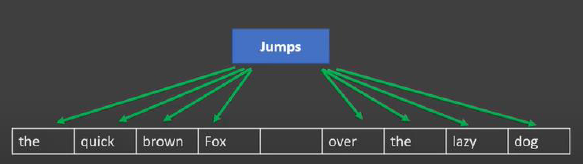
→ Sacar mejor esta imagen. **Modelo de Red Neuronal utilizado para entrenar embeddings**

****

**Entrenamiento mediante CBOW vs Skip-gram**

****

**CBOW – Predecimos la palabra “target” en base al contexto**

****

**Skip Gram – Predecimos el contexto en base a la palabra “target”.**

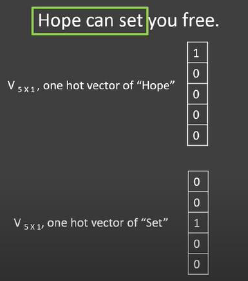
**2.3.4.1-Funcionamiento Word2Vec para obtener nuestros Word Embeddings.**

En nuestro 1er análisis utilizaremos CBOW como entrenamiento de nuestro modelo Word2Vec para obtener nuestros Word Embeddings. Pero, como vimos, también está el entrenamiento mediante Skip Grams. Veamos un ejemplo de cómo obtener nuestros Word Embeddings:

* **Mediante CBOW:**

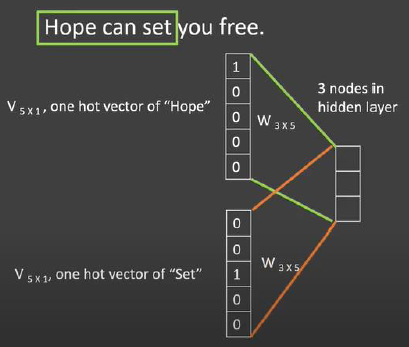
Consideremos que tenemos un vocabulario simple (“Hope can set you free”) con el que entrenaremos nuestra red neuronal (propia del modelo Word2Vec).

**Paso 1.** Pasamos nuestro vocabulario a un “one hot vector” (donde tendrá un 1 en la posición que tiene la palabra en el texto y un 0 en las demás posiciones; y el tamaño del vector será igual al número de palabras de nuestro lenguaje -nuestro vocabulario-). Luego de esto seleccionamos un tamaño de ventana (en nuestro seleccionaremos 3). Como en CBOW tratamos de predecir la palabra en base al contexto, entonces usamos como entrada “hope” y “set” (contexto) para predecir la palabra target del medio (“can”).

****

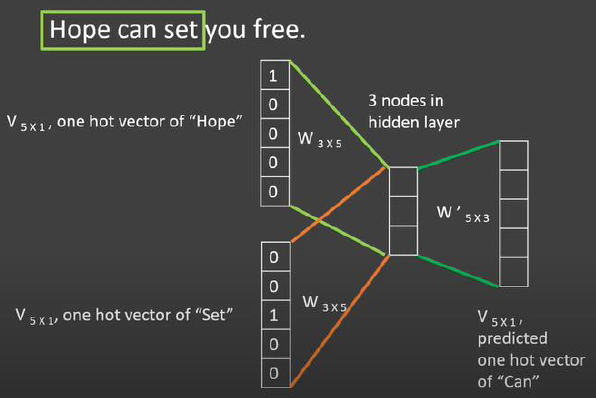
**Paso 1.**

**Paso 2.** Colocamos los vectores ‘one hot’ de “Hope” y de “Set” como entradas a nuestra red neuronal, la cual tendrá el 3 nodos en la capa oculta.

****

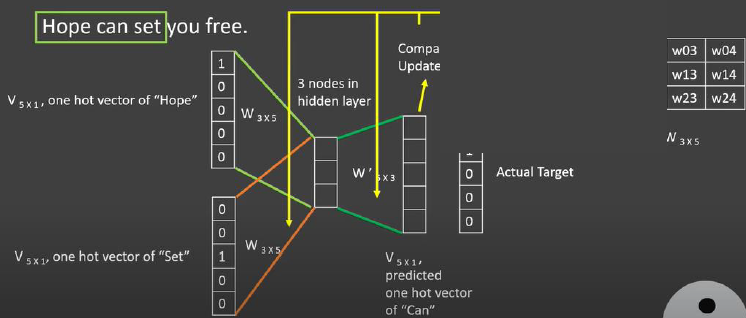
**Paso 2**

**Paso 3.** Entonces nuestra red neuronal tratará de predecir la palabra “can”. Estos valores pasarán por la función de la capa de salida SOFTMAX para obtener así probabilidades.

****

**Paso 3**

**Paso 4.** Dichas probabilidades obtenidas las comparamos con el vector “actual target” (que son los valores del vector “can”) y en base a esto actualizamos los pesos de nuestra red. Y luego de realizar iteraciones/epochs (otro parámetro a considerar) nuestra red neuronal tendrá una matriz con nuevos pesos para la 1ra iteración de la ventana de tamaño 3.

****

**→ Despues conseguir la imagen completaaaa**

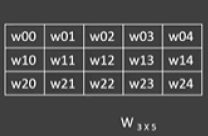
**Paso 4**

**Paso 5.** Luego continuamos con la 2da posición, hacemos lo mismo que vimos previamente, y luego lo mismo para la 3ra posición.

****

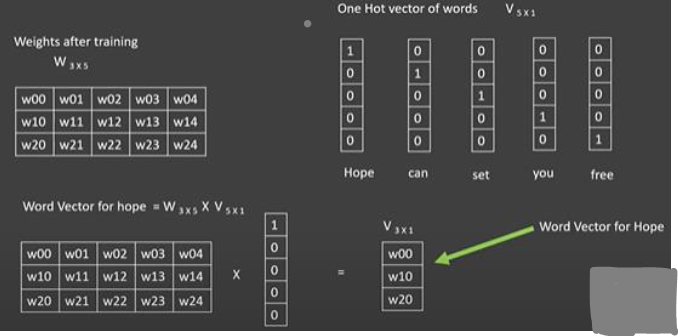
**Paso 5 (2da posición imagen izquierda; 3ra posición imagen derecha)**

**Paso 6.** Así, obtenemos esta matriz con los pesos de nuestra red neuronal luego del entrenamiento.

****

**Paso 6**

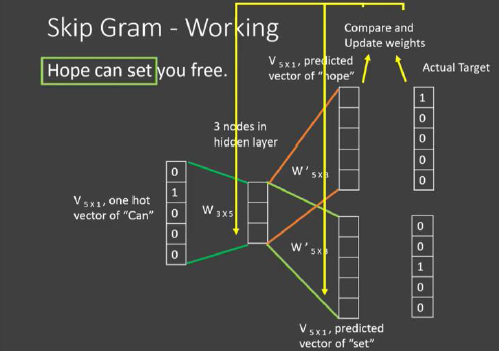
**Paso 7.** Luego, para obtener finalmente nuestros word embeddings para cada palabra, lo que hacemos es multiplicar la matriz obtenida en 6 por el vector ‘one hoy’ asociado a dicha palabra. En el caso de abajo observamos nuestro word embedding vector para la palabra “Hope”.

****

**Paso 7**

* **Mediante Skip Gram:**

En este caso, a diferencia de CBOW, tratamos de predecir el contexto en base a la palabra target. Por esto, nuestra entrada será el vector ‘one hot’ can y en base a esto predeciremos las palabras ‘hope’ y set’. Pero nuestro resultado será el mismo (matriz con los pesos asociados a nuestra red neuronal del paso 6- en CBOW).

****

**Skip Gram – Predecimos el contexto en base a la palabra “target”.**

**Para tener en cuenta:** En aplicaciones reales generalmente entrenamos nuestro texto con millones de palabras.

## 2.4-Cómo Clasificar Texto.

Para CLASIFICAR texto, lo que podemos hacer es:

1. Con nuestro corpus de texto ya pasado a n Vectores Embedding (1 por cada palabra de nuestro texto), podemos transformar estos N-embeddings en 1 solo mediante una vectorización de embeddings. Hay 2 opciones para esto:

* Promediar (AVERAGE) los embeddings y quedarnos con un único embedding que represente todo nuestro texto.
* Usamos “Doc2Vec” o “Sentence2Vec” que también nos va a procesar todas las palabras y nos va a dar un único embedding.

La diferencia de esta representación con la que se puede llegar a obtener de una representación de BoW, es que a diferencia de esta nos generará un vector de embedding mucho más compacto (de menos dimensiones que BoW).

2. Luego de esto, teniendo ya vectorizados con embeddings nuestros textos (o sea nuestro Corpus representado con 1 solo vector de embeddings) podemos utilizar distintos modelos de Deep Learning para procesar el texto en lenguaje natural y obtener dicha clasificación. Para esto tenemos distintas opciones:

* **CNN o CONV 1D (Redes Neunorales Convolucionales en 1 Dimensión):** permite aplicar una serie de N filtros con un cierto tamaño K de palabras. Aplica dichos filtros a todo el vocabulario, obtiene max poolings y a cada resultado le puede aplicar más convoluciones. De esta manera, se realiza una convolución de convoluciones de filtros. De esta manera se pueden reconocer si los textos tenían bi-gramas o tri-gramas, etc. y se empiezan a construir features más complejos en base a los textos. De la misma forma que una CNN sobre imágenes iba trabajando primero con bordes, luego con formas y demás... esto va trabajando primero con bigramas, luego con trigramas para entender semánticamente sentencias y demás, hasta llegar finalmente a una capa fully conected y finalmente a una SOFTMAX que nos dice a qué Clase pertenece nuestro texto.
* **CHAR-CONV 1D:**  es lo mismo que CONV 1D solo que envés de hacer los embeddings one hot a nivel de palabra se hacen a nivel de carácter. De esta manera tenemos un vector “a” por ej. de 100 posiciones (si hay 100 caracteres posibles) que tiene todos 0s y un 1 en alguna posición.
* **Bidireccional LSTM (B+- LSTM).**
* **Modelos de atención (Attention Models).**
* **RNNs.**
* **CHAR-RNNs.**
* **LSTM.**

La variante que funcione mejor depende del tipo de problema, del texto y del procesamiento que queramos hacer con ellos. En nuestro caso utilizaremos los modelos: ………………., los cuales detallaremos a continuación.

**2.4.1-Naive Bayes.**

**2.4.2-CNN.**

**2.4.3-LSTM.**

**2.4.4-Transformers.**

**2.4.5-BERT.**

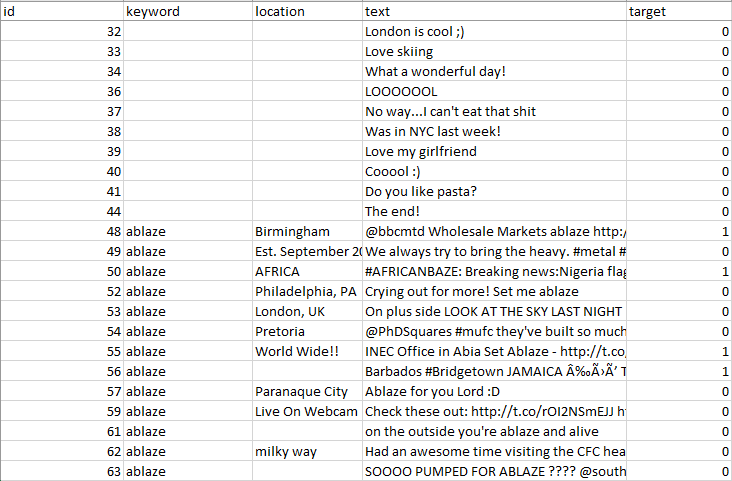
## 3- Desarrollo.

## 3.1-Data set.

El dataset que analizaremos en este Trabajo Práctico serán 2 archivos CSVs.

* ....csv: …………….
* Train.csv: …………….

Ambos datasets están disponibles en el siguiente link de la competencia de Kaggle: [https://www.kaggle.com/c/nlp-getting-started/data](https://www.kaggle.com/c/nlp-getting-started/data?select=train.csv). Y ambos cuentan con datos de tweets de distintas personas hechas en distintas ubicaciones(‘location’) que fueron o no tweets que tratan sobre desastres. Poseen 7613 filas y 5 columnas (VER), donde ada columna representa un feature de nuestros tweets. La diferencia entre ambos dataset, es que ‘train.csv’ tiene una columna ‘target’ que indica si el tweet verdaderamente habla sobre un desastre (‘1’) o si NO se trata de un desastre (‘0’).



***Imagen 1****. Data Set ‘.........csv’ (ejemplo primeros registros)*

Columnas de nuestro Data Set:

* id - identificador único para cada tweet.
* keyword - una palabra clave del tweet (puede no tener valor: Nan).
* location - la ubicación de donde el tweet fue enviado (puede no tener valor: Nan).
* text - texto del tweet.
* target - permite saber si el tweet trata sobre desastres (1) o no (0).

**3.2- Resultados y análisis de Tweets - TP1**

En el TP1 anterior, tuvimos la posibilidad de realizar un análisis exploratorio, obteniendo así distintas respuestas a nuestras preguntas mediante análisis estadísticos y visualizaciones:

1.¿Cuáles son las keywords (palabras claves) con mayor aparición en los tweets?

2.¿Varía la aparición de las keywords según el tipo de noticia (Target)?

3.¿Cómo influye si la keyword aparece en el texto del tweet?

4.¿Cómo están distribuidos los tweets según su keyword y su valor de target / veracidad?

5.¿Cuáles son las Top 30 ubicaciones con mayor cantidad de tweets?

6. ¿Varía la aparición de las keywords según el tipo de noticia (Target)?  
7.¿Cuál es la veracidad de los tweets con ubicación New York?

8.¿Cuáles son las longitudes de nuestros Tweets?

9.¿Cuáles son las longitudes de nuestros Tweets según su veracidad?

10.¿Cómo está distribuida la longitud de los tweets según su veracidad?

11.¿Cómo están distribuidos los tweets según la cantidad de caracteres de cada uno?

Este análisis exploratorio y construcción de visualizaciones, nos ayudó a entender los datos de nuestro dataset, buscar características e irregularidades. En esta 2da etapa, ya teniendo nuestro dataset analizado, lo que haremos en el desarrollo es utilizar distintos algoritmos de NLP para predecir cada tweet (si fue o no una catástrofe verídica).

## 3.3-Preprocesamiento de Texto.

VER SI ACÁ LO TOKENIZAMOS, SACAMOS LAS STOP WORDS Y removemos palabras comunes que no son requeridas O NO (estas palabras las colocamos en un archivo “commons.txt” donde colocamos palabras comunes que NO son necesarias para el análisis…. PERO CREO QUE ACÁ NO, ACÁ TODAS LAS PALABRAS IMPORTAN).

→ Imagen del texto antes de tokenizar y después de tokenizarlo a aplicarle tood eso.

Tambien ver si unimos las palabras bigramas → como “machine\_learning” o no.

**3.2-Obtención de nuestro Modelo y Predicciones de Tweets.**

**→ VER SI DIVIDIR EN 2 : OBTENCIÓN DEL MODELO (CON WORD EMBEDDINGS PRE-ENTRENADOS O ENTRENADOS CON NUESTROS TEXTOS SIMPLEMENTE) Y DESPUÉS PREDICCIONES O VER SI ES TODO “LO MISMO”.**

**→ Y Ver si acá utilizamos word2vec para crear los word embeddings con nuestros textos… y en mi informe tengo explicado esto bien.**

**3.2.1- VER METODO MAS BASICO → BoW y NAIVE BAYES?**

**3.2.2-VER USANDO EMBEDDINGS ENTRENADOS CON NUESTROS TEXTOS y predicciones con… VER.**

**3.2.3-USANDO EMBEDDINGS YA ENTRENADOS y predicciones con… VER.**

**3.2.4- OTRO METODO → CNN Y LSTM SEPARADOS?**

**3.2.5- OTRO MÉTODO → CNN Y LSTM JUNTOS?**

**3.2.6- OTRO METODO ¿TRANSFORMER?**

**3.2.7- BERT.**

**3.3-Resultados finales.**

# 4-Conclusión.

**5-Mejoras.**

# 

# 6-Bibliografía.

* **Apunte de la materia:** [**https://piazza.com/class\_profile/get\_resource/k7s41iiajq271y/k7s577n4h1g14s**](https://piazza.com/class_profile/get_resource/k7s41iiajq271y/k7s577n4h1g14s)
* [**https://medium.com/@gruizdevilla/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f**](https://medium.com/@gruizdevilla/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f)
* [**https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/**](https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/)