



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital  
Universitat Politècnica de València

## **Identificación de perfiles divulgadores de noticias falsas en Twitter**

Aplicaciones de la Lingüística Computacional

*Autor:* Jacobo López Fernández  
Juan Antonio López Ramírez

Curso 2019-2020



# Índice general

---

|  |   |
|--|---|
| Índice general   | 1 |
| 1 Introducción   | 2 |
| 2 Preproceso, limpieza y vectorización de los datos          | 3 |
| 3 Modelos utilizados y resultados                            | 4 |
| 3.1 Ajuste de parámetros para el modelo en español . . . . . | 6 |
| 3.2 Ajuste de parámetros para el modelo en inglés . . . . .  | 6 |
| 4 Evaluación en TIRA   | 8 |

---

# CAPÍTULO 1

## Introducción

---

Este trabajo consiste en la realización de un sistema para el análisis e identificación de perfiles de Twitter que sean divulgadores de noticias falsas.

La tarea recibe el nombre de *Profiling Fake News Spreaders on Twitter*<sup>1</sup>, y está recogida dentro de una serie de eventos científicos y tareas compartidas sobre texto digital forense y estilometría, denominada PAN. Concretamente, se trata de una tarea de análisis de autoría.

El conjunto de datos proporcionado consiste en dos carpetas por idioma (una para el inglés y otra para el español), donde cada carpeta tiene:

- Un XML por cada usuario de Twitter con 100 tweets. En total, hay 300 XML (300 autores), y el nombre del XML se corresponde con el identificador asignado para dicho autor.
- Un fichero de texto con el identificador del autor y la etiqueta de clase de cada uno de ellos.

De esta forma, el objetivo de nuestro sistema es, tomando como entrada los datos mencionados, generar para cada documento del dataset un archivo XML correspondiente en el que se recoja el identificador del autor, el idioma y la etiqueta de la clase a la cual pertenece.

La evaluación de nuestro sistema consistirá en la precisión o *accuracy* a la hora de clasificar. Para cada idioma, se calcularán las precisiones individuales al discriminar entre las dos clases y, finalmente, se hará un promedio de los valores de precisión por idioma para obtener la clasificación final.

---

<sup>1</sup><https://pan.webis.de/clef20/pan20-web/author-profiling.html>

---

## CAPÍTULO 2

# Preproceso, limpieza y vectorización de los datos

---

Lo primero que realiza nuestro *script* es un procesamiento de los datos para transformarlos y que obtengan la forma necesaria para poder pasárselos a los modelos que posteriormente usaremos.

Se han cargado los dos ficheros *truth.txt* del inglés y el español para guardar en dos listas de *Python* diferentes las etiquetas de cada clase y de cada idioma.

Al mismo tiempo, como las etiquetas están relacionadas con el identificador del autor y este a su vez es el nombre de su XML correspondiente, guardamos una lista de identificadores siguiendo el mismo orden que las etiquetas de clase.

Una vez hecho esto, recorreremos la lista de identificadores, cargando de cada XML los tweets que contiene. Como las muestras para entrenamiento y validación han de consistir en toda la historia del autor, cada una de las muestras ha de contener todos los tweets de cada usuario. Por ello, se procede uniendo en una misma cadena de caracteres los 100 tweets de cada autor, separando cada tweet por un espacio en blanco. Este proceso se repite para los 300 autores en inglés y los otros 300 en español.

Cuando ya tenemos esto resuelto, procedemos a vectorizar las muestras. Para ello, se ha hecho uso de *Scikit-learn*, una biblioteca para aprendizaje automático. Dicha librería nos proporciona el vectorizador *CountVectorizer*<sup>1</sup>, que es el que hemos utilizado.

De forma paralela, el tokenizador que se le ha pasado al vectorizador ha sido el *casual\_tokenize*, proporcionado por la librería de procesamiento del lenguaje natural NLTK<sup>2</sup>. Consideramos que este tokenizador era el adecuado para esta tarea debido a que es un tokenizador de tweets, es decir, tiene en cuenta la semántica empleada por los mensajes de esta red social.

Así, el vectorizador transforma el conjunto de muestras, guardando los resultados en dos matrices (una para cada idioma).

---

<sup>1</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.CountVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

<sup>2</sup>[https://www.nltk.org/\\_modules/nltk/tokenize/casual.html](https://www.nltk.org/_modules/nltk/tokenize/casual.html)

---

## CAPÍTULO 3

# Modelos utilizados y resultados

---

Cuando ya tenemos el texto preprocesado, hemos probado distintos modelos de clasificación para encontrar el que mejor rendimiento nos proporcione para nuestra tarea. Hemos tenido en cuenta, como puntuación mas relevante, el *accuracy* que se obtiene al evaluar el conjunto de entrenamiento con validación cruzada usando 10 bloques. De esta manera, el *accuracy* general se calcula haciendo una media de las 10 precisiones obtenidas.

Los modelos utilizados han sido proporcionados, una vez más, por la librería *Scikit-Learn*, y consisten en algoritmos de Machine Learning, como pueden ser el de vecinos más cercanos, vectores soporte, descenso por gradiente estocástico, etc. Además de estos, también hemos hecho experimentos con *Keras* para implementar redes recurrentes y comprobar si mejorábamos los resultados.

En este último caso, usamos el entorno de trabajo de *Google Collaboratory*, donde se nos proporcionan unos recursos de aceleración por GPU que sirven de ayuda a la hora de entrenar modelos de redes neuronales, y que si se ejecutasen en un entorno local el tiempo de cómputo sería mucho más elevado.

Además, se han tenido que cargar *word embeddings* preentrenados que nos ayudasen a codificar las palabras y vincularlas a vectores de números reales que poder proporcionar a la red neuronal de nuestro sistema.

Para el caso del español, se cargaron los *embeddings* del dataset *Spanish Billion Words Corpus and Embeddings*<sup>1</sup>, que se entrenaron usando *word2vec*, constan de alrededor de 1 millón de palabras, donde cada una está codificada en un vector de tamaño 300.

Para el caso del inglés, usamos los *embeddings* preentrenados con *GloVe* de Stanford y recogidos por Laurence Moroney<sup>2</sup>, que constan de 6 billones de palabras, cada una está codificada en vectores de tamaño 100.

Una vez cargados los embeddings, se entrena el modelo de redes recurrentes con LSTM, cuyo diseño se puede apreciar en la figura 3.1.

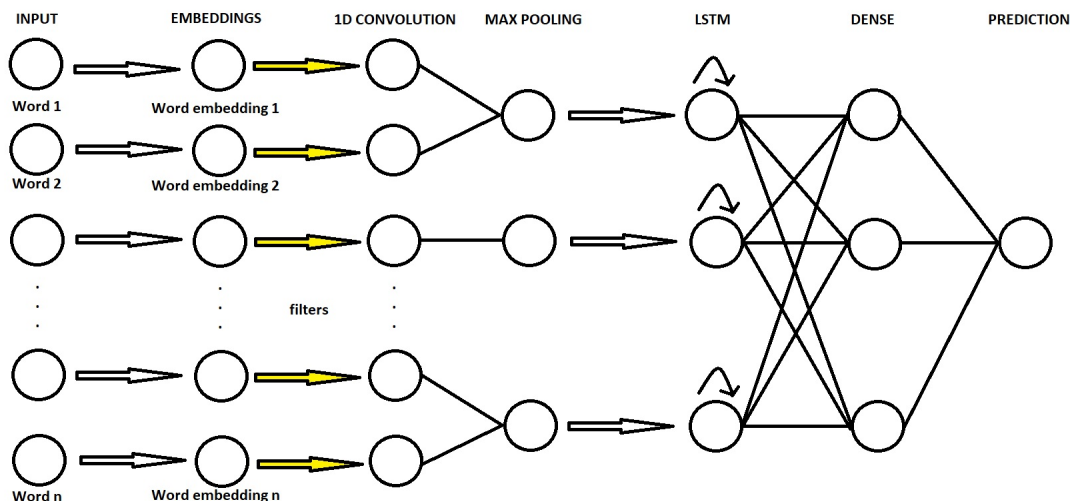
Los mejores resultados obtenidos utilizando este sistema fueron un *accuracy* de un 75 % para el español y un 61 % para el inglés. Como estos resultados no fueron los deseados debido a que esperábamos unos porcentajes más elevados, decidimos utilizar otros modelos.

Como hemos comentado antes, el resto de modelos que se usaron vienen implementados en la librería de *Scikit-Learn* y decidimos usar 7 distintos, aunque uno de ellos es

---

<sup>1</sup><https://www.kaggle.com/rtatman/pretrained-word-vectors-for-spanish>

<sup>2</sup><https://storage.googleapis.com/laurencemoroney-blog.appspot.com/glove.6B.100d.txt>



**Figura 3.1:** Diseño de la red neuronal recurrente con convoluciones y LSTM.

una versión distinta de otro, como puede ser el de vectores soporte<sup>3</sup>, en el que se utiliza su versión original y la versión lineal.

Teniendo en cuenta la precisión media alcanzada a partir de los conjuntos de validación cruzada, los resultados obtenidos han sido los siguientes:

| Modelo                             | Accuracy en inglés (%) | Accuracy en español (%) |
|------------------------------------|------------------------|-------------------------|
| Vectores soporte                   | 63                     | 74                      |
| Vectores soporte (versión lineal)  | 64                     | <b>83</b>               |
| Naive Bayes Gaussiano              | 64                     | 70                      |
| Gradient Boosting                  | <b>71</b>              | 76                      |
| Descenso por gradiente estocástico | 67                     | 78                      |
| Vecinos más cercanos               | 59                     | 74                      |
| Perceptrón Multicapa               | 65                     | 82                      |
| Red Neuronal Recurrente (LSTM)     | 60                     | 66                      |

Como podemos apreciar, todas las ejecuciones se han realizado sin alterar las tareas de tokenización y vectorización (a excepción de la LSTM, en la que usábamos *word embeddings*), para poder observar claramente las variaciones de resultados al aplicar distintos tipos de clasificadores. De los distintos clasificadores, los únicos que han visto modificados sus hiperparámetros han sido el de vectores soporte (lineal y no lineal), al que se le ha añadido un parámetro  $C=1$ ; además del Perceptrón Multicapa, al que hemos modificado aspectos como el optimizador, las iteraciones o el número de capas ocultas para maximizar el resultado.

Aun así, como el mejor modelo para el español ha sido el clasificador de vectores soporte lineal, se ha realizado el ajuste de los hiperparámetros a partir de él. De la misma forma, se ajustaron los parámetros del *Gradient Boosting*<sup>4</sup> para optimizar su resultado inicial para el inglés.

<sup>3</sup><https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496>

<sup>4</sup><https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af>

### 3.1 Ajuste de parámetros para el modelo en español

---

Los diferentes parámetros del modelo de vectores soporte lineal son:

- El parámetro  $C$ , para la penalización de los errores. Hemos probado distintos valores (1000, 100, 1 y 0.01).
- El umbral de tolerancia para determinar la convergencia del algoritmo. Los valores que se han probado son 0.1, 0.01 y 0.001.
- La función de coste o *loss*. Por defecto, hemos usado siempre la función *hinge*, debido a que la variación de esta no supone una repercusión significativa en la métrica de precisión.
- El tipo de normalización aplicada a la penalización.
- El número máximo de iteraciones antes de parar el entrenamiento.

Dado que los hiperparámetros más importantes de las máquinas de vectores soporte son la tolerancia y  $C$ , se ha decidido variar estos dos, dejando como función de penalización la función *hinge* y  $l_2$  como la normalización de la penalización. Además, a pesar de no lograr la convergencia del modelo, si que se obtiene el mejor resultado, estableciendo un valor concreto para el número máximo de iteraciones (100).

Los valores de  $C$  probados han sido 1000, 100, 1 y 0.01; mientras que los de la tolerancia han sido 0.1, 0.01 y 0.001.

Los resultados obtenidos, ajustando dichos parámetros, oscilan entre el 83 % que obteníamos previamente, y el 84 %, que a pesar de ser mínima, supone una mejora.

Dado que el mejor resultado es de 84 % en repetidas ocasiones para distintas configuraciones, escogemos aquella que menos modifique los parámetros establecidos por defecto.

Por tanto, el modelo de clasificador basado en máquinas de vectores soporte lineal con el que se realizará la estimación sobre el conjunto de test en TIRA se ha configurado con los parámetros:

- $C = 100$
- $tol = 0.01$
- $loss = 'hinge'$
- $penalty = 'l_2'$
- $max\_iter = 100$

### 3.2 Ajuste de parámetros para el modelo en inglés

---

Los parámetros del modelo de *Gradient Boosting* más destacables son:

- La función de pérdida, que puede ser 'deviance' si se trata de una regresión logística o 'exponential' para utilizar el algoritmo AdaBoost<sup>5</sup>.

---

<sup>5</sup><https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c>



- El *learning rate*, que en nuestro caso tomará valores entre 0.1 y 0.001.
- El *n\_estimators*, que es el número de etapas que realiza el algoritmo. Debido a que si en el algoritmo se usa un número de árboles elevado puede haber *overfitting* y, además, es intratable computacionalmente dar a este parámetro valores excesivamente altos, se tomarán como valores 100 y 250.

De esta forma, los resultados obtenidos modificando dichos parámetros han sido los siguientes:

| Loss        | learning rate | n_estimators | Accuracy (%) |
|-------------|---------------|--------------|--------------|
| deviance    | 0.1           | 100          | 71           |
| deviance    | 0.1           | 250          | 71           |
| deviance    | 0.01          | 100          | 70           |
| deviance    | 0.01          | 250          | <b>73</b>    |
| deviance    | 0.001         | 100          | 65           |
| deviance    | 0.001         | 250          | 68           |
| exponential | 0.1           | 100          | 72           |
| exponential | 0.1           | 250          | 71           |
| exponential | 0.01          | 100          | 70           |
| exponential | 0.01          | 250          | <b>73</b>    |
| exponential | 0.001         | 100          | 65           |
| exponential | 0.001         | 250          | 68           |

A la vista de estos se puede observar que influye positivamente reducir el factor de aprendizaje a 0.01 y aumentar el número de etapas del algoritmo a 250. Por otra parte, el hecho de modificar la función de pérdida no afecta significativamente al resultado obtenido, por lo que se ha decidido dejar este parámetro con la configuración por defecto, es decir, utilizando la regresión logística.

Así, el modelo de clasificador basado en *Gradient Boosting* con el que se realizará la estimación sobre el conjunto de test en TIRA se ha configurado con los parámetros:

- `loss = 'deviance'`
- `learning_rate = 0.01`
- `n_estimators = 250`

---

## CAPÍTULO 4

# Evaluación en TIRA

---

Para evaluar los modelos entrenados, se han subido a la máquina virtual junto con un script en python. A dicho programa se le pasan como argumentos el dataset de entrada, sobre el que los modelos en inglés y español (ambos con su respectivo vectorizador y clasificador ya entrenados) realizarán las predicciones; y un directorio de salida, donde se almacenarán los XML generados por nuestro sistema.

Así, utilizando el dataset de *training* (que previamente habíamos utilizado para realizar la validación cruzada y el entrenamiento de nuestro modelo), los resultados que obtuvimos fueron un **52.33 %** de precisión en español y un **98 %** en inglés (véase la figura 4.1).

```
measure{
  key:"lang"
  value:"es"
}
measure{
  key:"type"
  value:"0.5233"
}
measure{
  key:"lang"
  value:"en"
}
measure{
  key:"type"
  value:"0.9800"
}
```

**Figura 4.1:** Resultados obtenidos tras la primera ejecución en TIRA con el dataset de *training*.

Posteriormente, se hizo lo propio con el dataset de *test*, para el que se obtuvo una puntuación de **52.50 %** en español y **68.50 %** en inglés (véase la figura 4.2).

Debido a los bajos resultados obtenidos para la tarea en español, se procedió a implementar algunos cambios en el modelo para tratar de mejorar el *accuracy*.

Estos consistieron en sustituir el vectorizador, del *CountVectorizer* que se estaba utilizando hasta el momento, por el *TfidfVectorizer*<sup>1</sup>, ya que el primero solo cuenta el número de veces que aparece una palabra en un documento, mientras que el segundo ayuda a

---

<sup>1</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

```

measure{
  key:"lang"
  value:"es"
}
measure{
  key:"type"
  value:"0.5250"
}
measure{
  key:"lang"
  value:"en"
}
measure{
  key:"type"
  value:"0.6850"
}

```

**Figura 4.2:** Resultados obtenidos tras la primera ejecución en TIRA con el dataset de *test*.

lidar con las palabras más habituales y penaliza las menos comunes, ya que pondera los recuentos de palabras en función de la frecuencia con la que aparecen en los documentos.

Además, se modificó el modelo del lenguaje, en el que se estaban utilizando unigramas, sustituyéndolo por bigramas y trigramas (con el parámetro *ngram\_range*; además de procesar las tildes de las palabras (con el parámetro *strip\_accents*). Por último, mediante el parámetro *max\_features* con un valor de 1000, se construyó un vocabulario que solo considera las 1000 máximas características, ordenadas por frecuencia de término, en todo el corpus.

Una vez aplicados estos cambios para el español, se consiguió superar el **52.33** para el dataset de *training* hasta alcanzar un **99.67** (véase la figura 4.3), por lo que el modelo que se ha escogido como última opción para el español sigue siendo el *linearSVC* con los mismos parámetros que se utilizaron en un principio, pero con los cambios previamente descritos para el vectorizador. Con respecto al modelo en inglés, no se realizó ningún cambio al modelo de *Gradient Boosting*.

























```

measure{
  key:"lang"
  value:"es"
}
measure{
  key:"type"
  value:"0.9967"
}
measure{
  key:"lang"
  value:"en"
}
measure{
  key:"type"
  value:"0.9800"
}

```

**Figura 4.3:** Resultados obtenidos tras la segunda ejecución en TIRA con el dataset de *training*, después de modificar el vectorizador.

Finalmente, se realizó la última evaluación de los modelos, como se puede apreciar en la figura 4.4, donde la última ejecución (2020-05-31-17-52-34), es nuestra entrega final, cuyos resultados son un *accuracy* de **73.50** en español y **68.50** en inglés, como se puede observar en la figura 4.5.

| Runs        |                     |  |                     |          |        |   |
|-------------|---------------------|--|---------------------|----------|--------|---|
| Software    | Run                 | Input dataset                                      | Input run           | Runtime  | Size   | Actions   |
| evaluator21 | 2020-05-31-17-52-34 | pan20-author-profiling-test-dataset-2020-02-23     | 2020-05-31-17-44-19 | hidden   | hidden |    |
| software1   | 2020-05-31-17-44-19 | pan20-author-profiling-test-dataset-2020-02-23     | none                | hidden   | hidden |    |
| evaluator21 | 2020-05-31-17-30-03 | pan20-author-profiling-training-dataset-2020-02-23 | 2020-05-31-17-20-22 | 00:00:01 | 2.5K   |    |
| software1   | 2020-05-31-17-20-22 | pan20-author-profiling-training-dataset-2020-02-23 | none                | 00:00:41 | 344K   |    |
| evaluator21 | 2020-05-14-15-27-46 | pan20-author-profiling-test-dataset-2020-02-23     | 2020-05-14-15-21-25 | 00:00:01 | 2.5K   |    |
| software1   | 2020-05-14-15-21-25 | pan20-author-profiling-test-dataset-2020-02-23     | none                | 00:00:30 | 231K   |    |
| evaluator21 | 2020-05-14-15-17-19 | pan20-author-profiling-training-dataset-2020-02-23 | 2020-05-14-15-07-51 | 00:00:01 | 2.5K   |    |
| software1   | 2020-05-14-15-07-51 | pan20-author-profiling-training-dataset-2020-02-23 | none                | 00:00:39 | 344K   |    |

**Figura 4.4:** Ejecuciones realizadas en TIRA.

```

measure{
  key: "lang"
  value: "es"
}
measure{
  key: "type"
  value: "0.7350"
}
measure{
  key: "lang"
  value: "en"
}
measure{
  key: "type"
  value: "0.6850"
}

```

**Figura 4.5:** Resultados finales obtenidos en TIRA con el dataset de *test*.