




Organización de Datos 75.06/95.58  
Trabajo Práctico N° 2

Grupo 21

Apellido y Nombre	Padrón	Correo electrónico
Nicolas Farfan	97261	niko.f2@gmail.com

Github: 

<https://github.com/NicolasRFL/OrgaDatosTP2>

# Índice

<b>1. Introducción</b>	<b>2</b>
1.1. Hipótesis de Trabajo . . . . .	2
1.2. Información obtenida del TP 1 . . . . .	2
<b>2. Análisis</b>	<b>3</b>
2.1. Modelos de machine Learning . . . . .	3
2.1.1. Árboles de decisión . . . . .	3
2.1.2. RandomForest . . . . .	4
2.1.3. XGBoost . . . . .	5
2.1.4. XGBoost- TF Idf . . . . .	5
2.1.5. CatBoost . . . . .	6
2.2. Redes neuronales . . . . .	7
2.2.1. Logistic Regression . . . . .	8
2.2.2. SVM-regression . . . . .	8
2.2.3. GloVe Keras . . . . .	8
2.2.4. Roberta Classifier Model . . . . .	9
2.2.5. SimpleTransformers Roberta . . . . .	10
2.3. Modelos de procesamiento de texto . . . . .	10
2.4. Modelo elegido . . . . .	11
<b>3. Conclusiones</b>	<b>11</b>

# 1. Introducción

## 1.1. Hipótesis de Trabajo

El siguiente trabajo práctico consiste en determinar si un tweet es verdadero o falso, cuando solo se conoce el cuerpo del tweet, un keyword que identifica diversos desastres asociados a los tweets y la ubicación desde donde se envió el tweet. Para ello fue necesaria la creación de un modelo que permita extraer información relevante y determinar así la veracidad de los tweets.

Se uso pandas de python para procesar el archivo que contiene la información y crear nuevos features a partir de dicha información. Luego se implementaron diversos modelos de machine learning para analizar los features y crear una hipótesis de si un tweet informa sobre una catástrofe verdadera o falsa.

## 1.2. Información obtenida del TP 1

En el trabajo práctico anterior se hicieron hipótesis sobre que aspectos de la información dada podría afectar la veracidad de un tweet. Durante la creación del presente trabajo practico se tomo la decisión de crear los siguientes features:

- Un feature binario que determina si la ubicación desde donde se tweetea es un lugar geográficamente valido o no.
- Un feature que determina la longitud de un tweet.
- Un feature que cuenta la cantidad de hashtags de un tweet.
- Un feature que determina la cantidad de gente mencionada en un tweet (alguien mencionado en un tweet se denota por un @ y un nombre de usuario posteriormente).
- Un feature que clasifica según cual keyword se encuentra en un tweet (por lo general se usó mean hashing para este feature).

Luego se adapto estos features a cada algoritmo, se agregaron otros features (por ejemplo features que representan el texto en un vector) y se realizo feature engineering donde se vió provechoso, según el feature y el modelo.

## 2. Análisis

### 2.1. Modelos de machine Learning

Durante la realización de este trabajo practico se usaron distintos modelos de machine learning para determinar la veracidad de un tweet. Esos modelos se clasificaron en modelos de procesamiento de texto, arboles de decisión y redes neuronales según el tipo de features que utilizan y la performance con el set de datos dado.

#### 2.1.1. Árboles de decisión

Para los modelos de tipo arboles de decisión se decidió usar features categoricos, ya sea que se obtienen datos categoricos desde el set de datos dados, que se transformaron datos continuos o de tipo string en categoricos. Se usó binning (por ejemplo al usar la longitud de los tweets como feature, se dividió en bins del mismo tamaño para volver la información continua en categorica), binary encoding (se prefirió binary a one hot dado que crea menor cantidad de features y se disminuye la probabilidad de overfitting de los modelos) y corte basado en cuantiles (usando qcut de pandas). Se uso este criterio en la mayoría de los arboles dado que por cada feature el árbol decide a que rama saltar, si se convierte el feature en categórico la decisión se reduce a un problema binario, por ejemplo en el trabajo practico, si la longitud del tweet está en el bin 20-40 caracteres, se elige una rama, sino se elige la otra.

Por otra parte hay algunos árboles que no pueden recibir información continua o que si la reciben, pero a la hora de procesar los datos le dan demasiada importancia a los features continuos por sobre los demás features. Sin embargo hay árboles que no tienen problemas al procesar datos continuos pero crean categorías propias que son solo conocidas por el algoritmo y por lo tanto disminuyen el control del programador sobre el modelo.

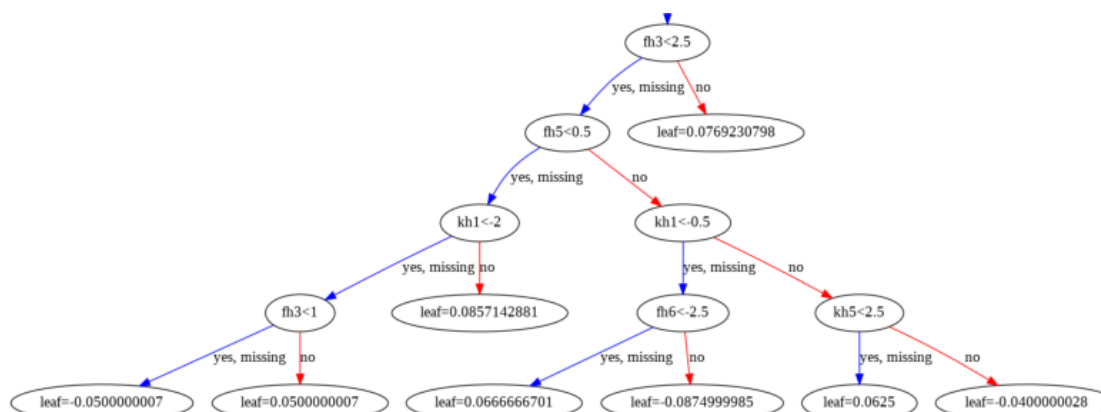


Figura 1: Corte de una rama de un arbol de XGBoost.

### 2.1.2. RandomForest

El primer algoritmo probado fue Random Forest de scikit learn. Se alcanzo una precisión de 0.65. Se eligio random forest porque es un modelo ligero y simple de implementar sobre un dataframe de pandas.

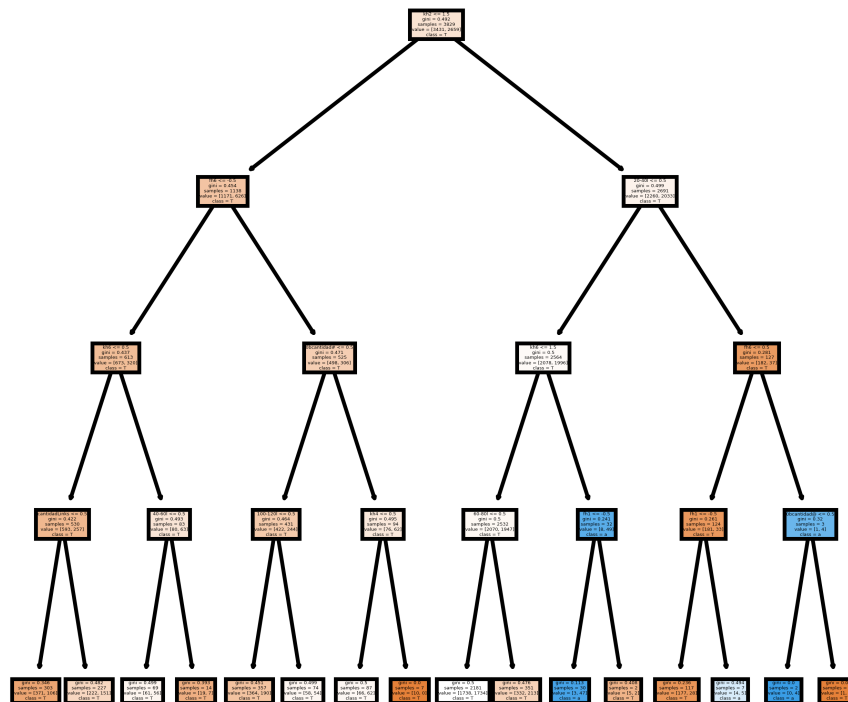


Figura 2: Árbol individual de decision Random Forest

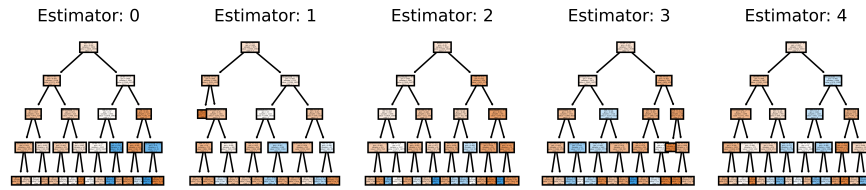


Figura 3: Random Forest

### 2.1.3. XGBoost

Se probó el algoritmo XGBoost, que tiene su propia librería XGB. Se eligió xgboost porque parecía una progresión natural desde random forest, siendo a su vez también simple de implementar y comprender sus resultados. Sin usar un algoritmo de procesamiento de texto, más allá del feature hashing proveído por scikit, se obtuvo una precisión de 0.71 en set de test y un error cuadrático medio de 0.44.

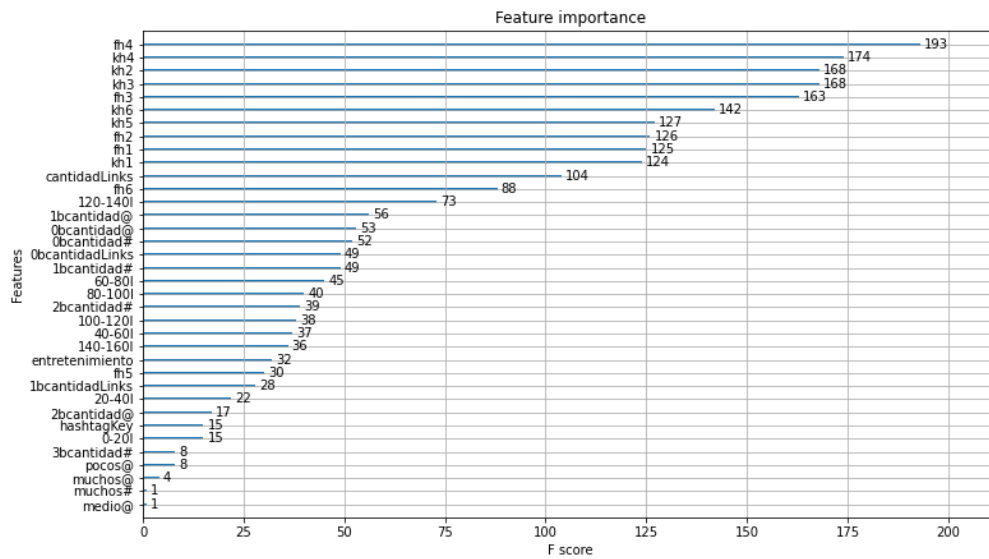


Figura 4: Gráfico de feature importance de xgboost

### 2.1.4. XGBoost- TF Idf

Se evaluó la performance de XGBoost usando TF Idf para procesar el texto, además de usar los features adicionales calculados para el intento anterior de XGBoost. Se alcanzó una precisión de 0.77.

### **2.1.5. CatBoost**

Por ultimo de los arboles de decisión se probó el algoritmo de CatBoost. Se obtuvo un error cuadrático medio de 0.436637 al evaluar el modelo y se alcanzo un accuracy del 0.72 al subir el modelo a kaggle.

## 2.2. Redes neuronales

Al usar redes neuronales fue necesario usar otro tipos de features dado que los features categoricos, si bien pueden ser interpretados por la mayoría de los algoritmos, no son el input óptimo que estos modelos puede recibir. Por esto, se usó frequency encoding y mean encoding, y antes de entregar los dataframe al modelo se uso un standard scaler proveido por scikit learn. El preprocesador standard scaler estandariza los features quitando la media del feature y haciendo que la varianza sea 1. Este preprocesamiento se hace para mejorar el comportamiento de los modelos estimadores de machine learning.

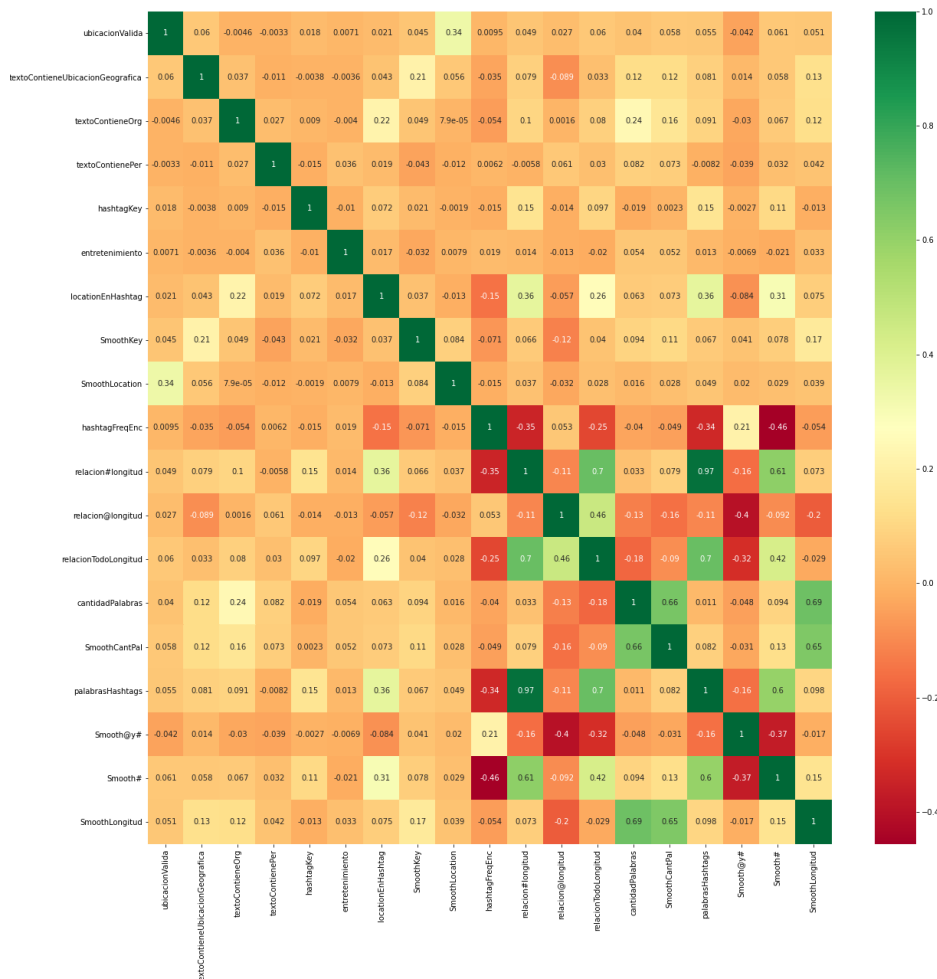


Figura 5: Heat map de features usados en regresión logística



### 2.2.1. Logistic Regression

El primer modelo de redes neuronales fue la regresión logística. Se obtuvieron resultados relativamente buenos, de 0.74 usando doc2vec para codificar el texto de los documentos pero features simples, que solo calculaban promedios y diferencias. Al realizar mean encoding con smoothing y frequency encoding se mejoró el resultado a 0.76.

### 2.2.2. SVM-regression

La red neuronal SVM SVC se eligió por ser simple y se considero una alternativa a la regresión logística dada la facilidad para trabajar con pandas. Se obtuvo una precisión del 0.77.

### 2.2.3. GloVe Keras

Glove es un algoritmo de aprendizaje no supervisado que permite obtener representación vectoriales de palabras. Se intentó crear varios tipos de redes neuronales, entre ellas una red neuronal convolucional, usando GloVe como una embedding layer de un modelo de keras, sin ingresar ningún otro tipo de feature. Desgraciadamente se tuvo poco éxito (solo logrando un 0.56 de accuracy comparando con los datos de Kaggle). Se puede atribuir la poca precisión del modelo a la falta de experiencia al trabajar con keras y la falta de preprocesamiento del texto.

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	1471500
lstm (LSTM)	(None, 100, 128)	117248
lstm_1 (LSTM)	(None, 64)	49408
dense (Dense)	(None, 64)	4160
dense_1 (Dense)	(None, 1)	65
Total params: 1,642,381		
Trainable params: 170,881		
Non-trainable params: 1,471,500		

Figura 6: Gráfico mostrando uno de los modelos creados con keras.

### 2.2.4. Roberta Classifier Model

Se creó con keras un modelo de clasificación usando Roberta del paquete transformers. De los modelos de tipo BERT se eligió ROBERTA porque distintas pruebas destacan su performance sobre BERT.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT <sub>LARGE</sub>	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet <sub>LARGE</sub>	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	<b>90.2/90.2</b>	<b>94.7</b>	<b>92.2</b>	<b>86.6</b>	<b>96.4</b>	<b>90.9</b>	<b>68.0</b>	<b>92.4</b>	<b>91.3</b>	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	<b>90.7</b>	83.5	95.2	92.6	<b>68.6</b>	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	<b>96.8</b>	<b>93.0</b>	67.8	91.6	<b>90.4</b>	88.4
RoBERTa	<b>90.8/90.2</b>	<b>98.9</b>	90.2	<b>88.2</b>	96.7	92.3	67.8	<b>92.2</b>	89.0	<b>88.5</b>

Performance comparison from [RoBERTa](#).

Roberta toma el texto contenido en el tweet y lo procesa usando un embedding layer, para luego devolver una tupla que sirve para predecir el target. El modelo fue muy robusto pero complicado de construir, aún usando la documentación provista por el creador del modelo. Sin embargo, se destaca que se posee mayor control sobre el modelo que su alternativa simple transformers, se podría procesar el output del modelo usando keras, concatenando otro modelo para quizás incorporar los features calculados anteriormente. El clasificador devolvía una tupla a la cual fue necesario aplicar una transformación softmax para obtener la predicción.

```

191/191 [-----] - 123s 644ms/step - loss: 0.5086 - accuracy: 0.7606 - val_loss: 0.4347 - val_accuracy: 0.8227
Epoch 2/5
191/191 [-----] - 121s 631ms/step - loss: 0.4216 - accuracy: 0.8269 - val_loss: 0.4215 - val_accuracy: 0.8273
Epoch 3/5
191/191 [-----] - 121s 631ms/step - loss: 0.3617 - accuracy: 0.8571 - val_loss: 0.5516 - val_accuracy: 0.8129
Epoch 4/5
191/191 [-----] - 120s 631ms/step - loss: 0.3157 - accuracy: 0.8796 - val_loss: 0.4866 - val_accuracy: 0.8372
Epoch 5/5
191/191 [-----] - 121s 631ms/step - loss: 0.2730 - accuracy: 0.8977 - val_loss: 0.4559 - val_accuracy: 0.8234

```

Figura 7: Loss y precision según cada epoch

Aún así el modelo alcanza una precisión de 0.81 al predecir el target en Kaggle usando solo el texto de los tweets y el target del set de entrenamiento, sin features adicionales.

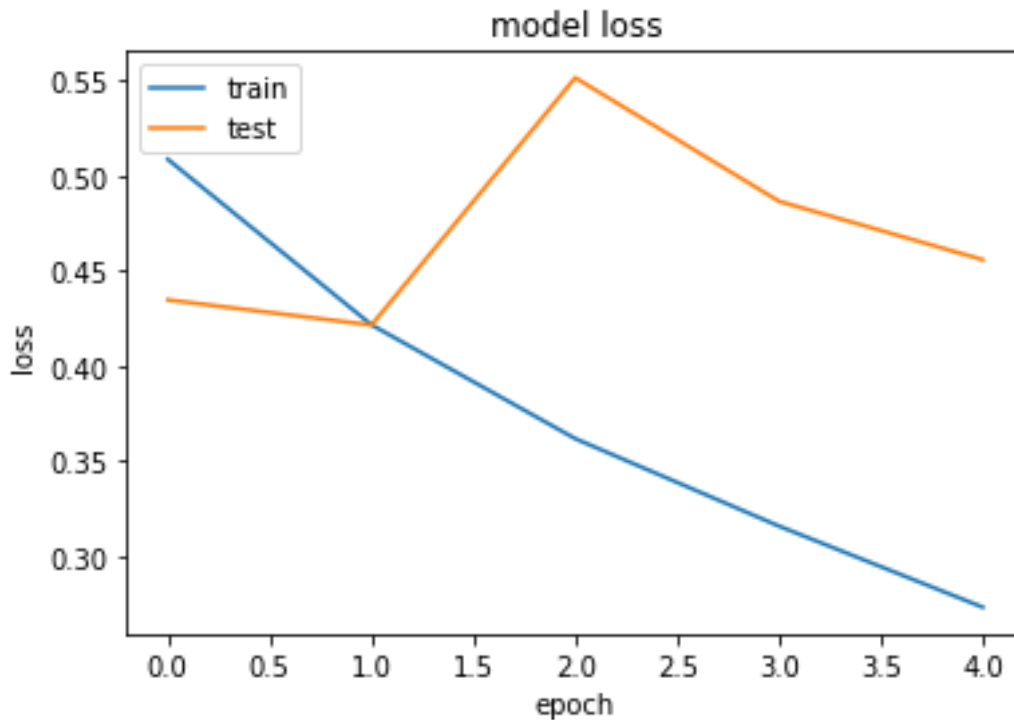


Figura 8: Model loss según epoch

### 2.2.5. SimpleTransformers Roberta

Simple Transformers usando Roberta fue el modelo que mayor precisión dio, alcanzando una precisión de 0.83 en Kaggle. Fue realmente simple de entrenar y obtener un resultado, dado que se puede usar como input un dataframe que contiene el texto y un label, y a la hora de predecir solo se necesita un vector que contiene cadenas de string.

Sin embargo dado que no se comprende mucho de cómo está implementado internamente el modelo de machine learning (solo se conoce que usa transformers), y dado la poca flexibilidad del modelo, no fue el preferido de este trabajo práctico. Se menciona la poca flexibilidad porque si uno deseara insertar features adicionales, no podría dado que el modelo devuelve solo el target.

## 2.3. Modelos de procesamiento de texto

Se probaron distintos modelos de procesamiento de texto, cada uno tiene distintas formas de obtener vectores para representar el texto. Se tuvo el mayor éxito usando Roberta, con su modelo de clasificación de keras.

Modelos usados:

- GloVe
- TF-IDF

- Doc2vec
- Roberta

## 2.4. Modelo elegido

El modelo elegido para predecir es el roberta classifier model, un modelo de keras layers, dado que a pesar que no fue el que mejor performance tubo, se puede ajustar el output para en algún futuro usar los otros features obtenidos en conjunto al modelo. De todas formas se señala que la performance obtenida fue del 0.81.

El modelo se puede encontrar en el siguiente link:

<https://github.com/NicolasRFL/OrgaDatosTP2/blob/master/RobertaReloaded.ipynb>.

## 3. Conclusiones

Dados los resultados obtenidos, se comprobó que el análisis de tipo NLP usando algoritmos de procesamiento de texto es mucho más poderoso que la simple extracción de features que uno puede hacer. Esto significa que probablemente los algoritmos de procesamiento de texto hacen un análisis lingüístico y captan distintas señales en el lenguaje y las asocian al target, en este caso la veracidad y falsedad de un tweet.

En este aspecto es necesario notar la importancia de algoritmos de tipo BERT, en este caso ROBERTA, que usan redes neuronales para reconocer patrones en el lenguaje. Durante la realización de este trabajo practico, se uso dos modelos con ROBERTA, uno usando Roberta como una embedding layer con un modelo en keras conocido, el otro usando un modelo de caja negra, donde uno simplemente ingresa el texto sobre el que se quiere realizar predicciones y el modelo obtiene un resultado. Sin embargo no se pudo probar que sucede si a un modelo de tipo BERT que acepte texto encodeado, ademas se ingresan los features obtenidos para los otros modelos probados anteriormente. Esto se podría lograr concatenando modelos de redes neuronales, uno que procese el texto con BERT y otro que procese los features y al final concatenarlos en un capa para obtener un resultado.

Finalmente es necesario destacar que el análisis obtenido anteriormente con features numéricos y categoricos extraídos del texto no fue en vano, dado que ayudo a comprender como interactuar con distintos modelos de machine learning, como responden los distintos modelos a los distintos tipos de features y hasta que precisión se puede esperar de cada modelo, dado un set de datos determinado.